1985

Design and implementation of a communications package to network a group of microcomputers

Narayan J. Kulkarni

ROCHESTER INSTITUTE OF TECHNOLOGY
College of Applied Science and Technology

Department of Computer Science

Design and Implementation of a Communications Package to Network a Group of Microcomputers

by

Narayan J. Kulkarni

A Thesis submitted to
The Faculty of the School of Computer Science and Technology
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science

Approved by

Prof. Margaret M. Reek

Dr. James E. Heliotis

Dr. Rajendra B. Nalavade

June 7, 1985

Design and Implementation of a Communications package
to Network a Group of Microcomputers

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I take this opportunity to thank Prof. Margaret Reek for her guidence and encouragement during the course of this thesis work. I appreciate the adjustments she made to her schedules to accommodate me with short notices.

I also thank Dr. Jim Heliotis for his guidance and suggestions.

My special thanks are also due to Dr. Rajendra Nalavade for his guidance and suggestions during the implementation of this project. It would have taken me a long time to overcome some of the implementation problems mentioned in Chapter 4 without his active help.

I also thank the faculty, staff and students of the Department of Industrial Engineering, RIT, for having accommodated me in their labs and for having allowed me to use the equipment in their labs in spite of its heavy utilization for their development work.

ABSTRACT

With the development of VLSI technology, more and more desktop computers are coming into the marketplace. A need is often felt to network these computers together to share their resources for a larger application where the data needs to be interchanged between these computers. This thesis work is an attempt to design and implement a communications package that allows various desk-top computers to communicate between each other by exchanging data between them.

The computers in this network send data to each other through a central message server. Physical medium of communication between these systems is standard RS-232C communication lines. The message server holds the message until the system to which the message was addressed requests for this message. Attempt is made to design this package based on the seven layer ISO Communications Model. The design is flexible enough to permit easy expansion of the network layer services for packet routing, congestion and flow control, etc.

KEYWORDS

Desktop Computers, Protocol, Network layer, Data Link layer, Physical layer, RS-232C, Polling, Message Server, Start-stop positive acknowledgement/retransmission protocol, ISO-OSI model, Packet, Frame, Computer Network, Communication Package, Checksum, Data encoding, Echo, Perkin Elmer-3220, DEC Professional-350, Tektronix-4113

With the development of VLSI, more and more desk-top computers are being introduced in the marketplace every day. It is very typical to find that an organization buys a computer system to meet a specific requirement; and when a need is felt for another system for another application, the organization ends up buying another totally different computer system which happens to be the most price-competitive system in the marketplace at that time. A situation soon arises where the organization ends up with a collection of different computer systems that run different applications, but cannot communicate with each other. This is not a problem as long as the applications being run on these systems are stand-alone and self-contained. However, in a situation where an application needs to be distributed over these multiple systems, these systems will not be of much use unless they can communicate with each other.

The Department of Industrial Engineering at RIT has a group of mini and microcomputer systems being used in their laboratories for research and development applications. Many of these systems are dedicated for specific research projects, and work as independent systems. However, need arises from time to time to network these systems together, because the capabilities of a single system are not sufficient for the scope of a single project, and data needs to be transferred from one system to another. Such situations are presently very difficult to deal with. One way, current-

ly in use, is uploading of files to a mainframe and then downloading them to the other system. Another way is transferring them through interchangeable storage media. Unfortunately, none of the systems accepts a floppy disk or any other medium used by another system. Moreover, file management system software on any one system is not compatible with that on any other system. The file transfer utility has to convert data from one format to another. What is really needed is a way of passing data from one system to another under program control in a 'real-time' environment.

This thesis work attempts to design and implement such a package. The intent is to provide the users with subroutines that will allow them to send/receive data to/from a specific system in the network. Various network topologies were considered, and a 'star' configuration was selected for implementation. The system located at the center of the star acts as a 'Message Server' for the systems that want to communicate with each other. A system that wants to send a message to another system sends it via the message server. The message server receives the message, stores it until the destination system requests it, and then sends it to the destination system. Routines were written and tested to enable user programs to send data to another system via the Message Server.

The design of the communication package was based on the ISO communications model. The physical layer consists of RS-232C lines between each system and the Message Server. The exchange of data is preceded by a 'handshake' between the communicating systems. Start-stop positive acknowledgement/retransmission protocol en-

sures that the frames are sent from one system to another in the desired order without loss of data. Error-detection is provided at the data-link layer to ensure error-free transmission of data.

The design and implementation details are presented in the following chapters. Recommendations regarding the possible uses of this package as well as possible enhancements to suit the growing network needs are discussed in the last chapter.

2.1 ISO MODEL FOR DATA COMMUNICATIONS

An attempt has been made in the design of this networking package to conform as much as possible to the protocol structure defined by the International Standards Organization (ISO). This model for Open System Interconnect (OSI) is supported by most of the active national and international organizations.

The purpose of the ISO reference model for Open System Interconnection is to provide for the co-ordination of standards development for the purpose of systems interconnection. The model presumes modularity of the networking support software based on the functionality. Each module takes the form of a layer in the model, and is responsible for providing selected networking services to the layer above. These services are provided by programs in that layer and through the services available from the layer below. In theory, any layer can be replaced by a new layer which provides the same services in a different way without affecting the user's perception of the network operation. In this way, networks can be tailored to specific needs while using common components.

The application programs that use the services of the network form the highest layer of this model. The lowest layer is the physical medium over which the data travels from one place to another. Fig.

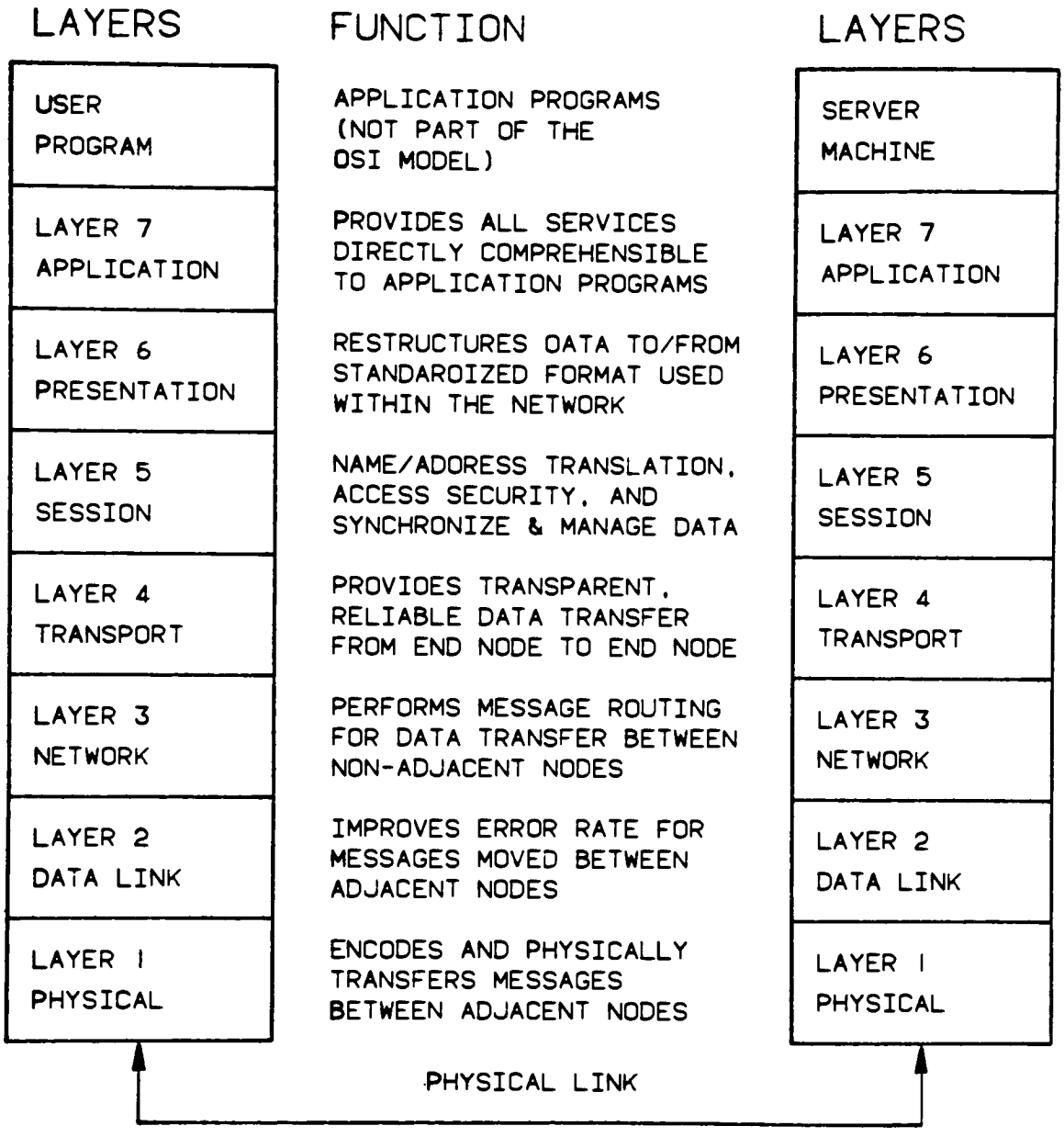


FIGURE 2.1 - OSI REFERENCE MODEL

2.1 gives an overview of the model with a description of the functionality of each of the layers. The model allows the implementation of any given layer in one or more sublayers in which the functionality of this layer is divided. A layer can be absent if the functionality is not desired.

The first two layers (Physical and Data Link Layers) deal with the communication between physically adjacent nodes. The protocols at these levels are consistent within each pair of adjacent nodes.

The network layer deals with the routing of messages in the form of smaller units (packets) between two or more communicating applications in the presence of the intermediate nodes.

Higher layers (Application, Presentation, Session, Transport and Network layers) are concerned with application level services, data format standardization, address translation, access security, and the end-to-end processing of the messages. Detailed explanation of the ISO model can be found in Reference 1.

In the implementation of this networking package, a facility is needed to exchange messages between various microcomputer systems as described in Chapter 3. This package does not offer any application level services, hence layers 4 and above are not considered for implementation in this package.

The 'Physical Layer' in this package is RS-232C communication lines. Based on the network topology selected, these lines will carry frames from one system to another.

The 'Data Link Layer' is designed to guarantee error-free transmission of a frame over the physical medium. The discussion of the communication protocol implemented by the data link layer follows this section.

The 'Network Layer' in this package is not a full blown network layer that performs the functions of static/dynamic routing, congestion control, crash recovery, etc. Due to the topology selected, the routing and congestion control functions are not applicable to this layer. This layer breaks the messages down to smaller packets and sends them over the data link layer. As explained in chapter 5, this layer can be expanded to meet the needs of anticipated expansion, by incorporating functions like congestion control, routing, sequencing of packets, detection of duplicates and missing packets, crash and recovery of intermediate nodes, etc. in the network layer.

2.2 COMMUNICATIONS PROTOCOLS

Protocols are established between systems in order to ensure an orderly exchange of information. It is a set of agreements between two communicating systems that incorporates the sequence of information exchange. Some consideration is given to the aspect of flow control by overlapping the transmissions of two systems to achieve a 'duplex' communication. Provisions need to be made in the protocol in order to prevent a 'fast sender' from overrunning a 'slow receiver'.

Figure 2.2 indicates the sequence of information exchange that

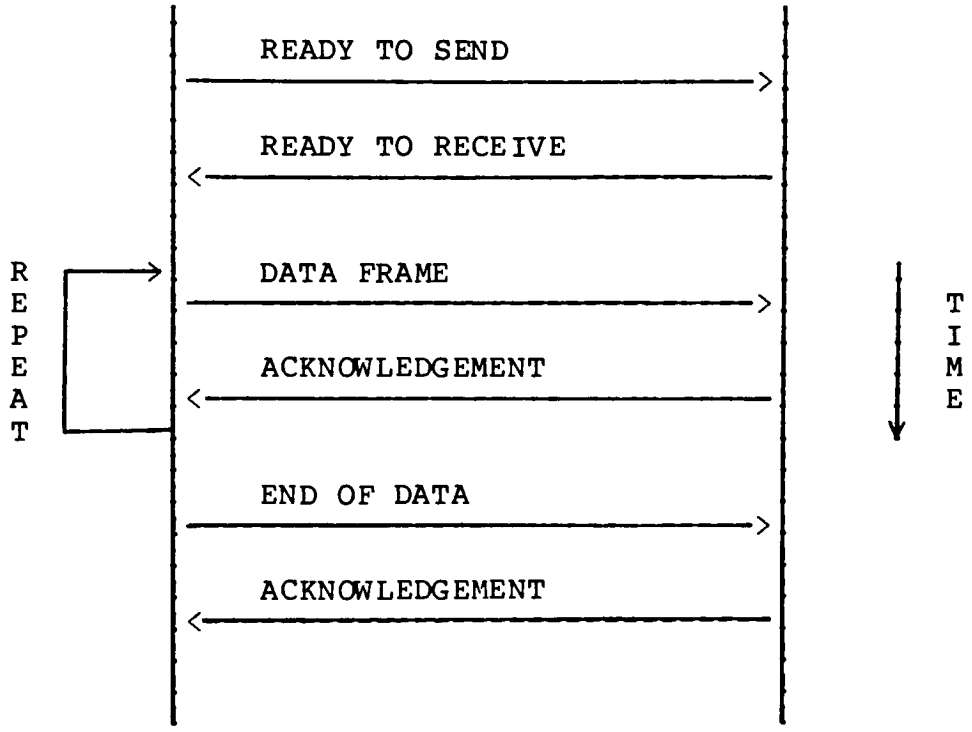


FIGURE 2.2 : A COMMUNICATION PROTOCOL MODEL

needs to be specified by a protocol.

Three steps indicated in the figure are as follows:

a. Exchange of control signals to establish communications.

b. Exchanges of data and acknowledgements. Retransmission of data if necessary.

c. Termination of the communications session.

Stop-and-wait protocols require the sender to wait for an acknowledgement after sending a data frame. The sender cannot send the next frame until a positive acknowledgement is received for the previous data frame.

Sliding window protocols allow some overlap between the transmission of the data frames by the sender and the transmission of the acknowledgements by the receiver. The sender can send multiple frames (depending on the size of the 'sending window') without having to wait for a positive acknowledgement for the data frame sent earlier. Receiver may receive the frames in a random order and may arrange them in proper order or may discard the frames that are out of sequence, depending upon the receiving window size.

The sliding window protocol utilizes the transmission lines more efficiently than the stop-and-wait protocol. At any given time, a number of message frames and acknowledgements may be traveling on the transmission lines. This makes it necessary for the software

on the sending as well as receiving system to be very complex, but achieves a larger throughput especially when the systems are separated from each other and communications delays are significant.

Since the systems under consideration are under the direct control of a single user, and are separated by only a few feet, the transmission delays in this case are almost non-existent. Therefore, a simplex protocol is selected for the implementation of the data link layer.

2.3 DESCRIPTION OF THE PROTOCOL

The data link layer is implemented by using a standard 'stop and wait positive acknowledgement/retransmission protocol'. The protocol establishes a communication between the sender and the receiver before the exchange of data begins. The communications are established as follows:

```
Send handshake request
wait for response
read response
if response=handshake acknowledgement
        begin transmission
else
        handshake failure, abort
endif
```

The transmission of data is accompanied by a checksum to assure the integrity of data. When the data is received by the receiver, the integrity of the data is checked by computing the checksum and comparing it with the checksum that accompanies the data. If the data received is correct, a positive acknowledgement is sent to the sender, otherwise a negative acknowledgement is sent.

The sender, after sending a data frame, waits for the receiver's response to verify that the frame was received by the receiver without any transmission errors. If a positive acknowledgement is received, then the next frame is sent, otherwise the same frame is re-transmitted. If a positive acknowledgement is not received after six attempts, the sender assumes that there is a problem with the physical layer, and aborts the transmission attempt. This communication protocol is illustrated in Fig. 2.3

Ideally, the sender is expected to set a timer and wait for a predetermined amount of time to receive a positive acknowledgement from the receiver for the data sent. The retransmission should begin if the sender gets a negative response, or fails to get a response before the timer fires; so that it does not have to wait indefinitely for receiving response from a system that may not be running. Unfortunately, this cannot be implemented as such due to the lack of timers on the hardware available. As a result, the system that transmits a handshake inquiry frame will keep indefinitely waiting for a handshake acknowledgement frame, and tie up the resources until the receiving system sends an acknowledge frame.

START-STOP POSITIVE ACKNOWLEDGEMENT/RETRANSMISSION PROTOCOL
(AS IMPLEMENTED)

```
send a packet
tries = 1
repeat {  read the response
          if response = nck
                send the packet again
                increment tries
          endif
       } until (response = ack or tries > 6)
```

FIGURE 2.3

START-STOP POSITIVE ACKNOWLEDGEMENT/RETRANSMISSION PROTOCOL
(IDEAL IMPLEMENTATION)

```
send a packet
set the timer
tries = 1
wait for an event
repeat { if event = framearrival
                if response = nak
                        retransmit the packet
                        increment tries
                endif
         else if event = timeout
                retransmit the packet
                increment tries
         endif
} until (response = ack or tries > 6)
```

FIGURE 2.4

The ideal communication protocol discussed above is illustrated in Fig. 2.4

2.4 TRANSMISSION:

The message to be transmitted is divided into smaller packets of equal length. A frame for transmission is constructed from the packet by adding a header and a trailer to it. The unit of transmission is 'frame'. There exist two different kinds of frames - control frames and data frames.

Control Frame: A control frame carries a control signal. The possible control signals are listed below:

* Handshake frames
* Handshake acknowledgement frames
* Positive/negative acknowledgement frames
* Request frame to receive data
* No-message indicator frame
* End-of-transmission frame

The control frames have a fixed length, and are not accompanied by any header or trailer information. For the sake of simplicity, the control frames do not need to be acknowledged.

Data Frame: A data frame carries a data packet. The data packet is preceded by a header and succeeded by a trailer. The header contains the type of the frame, addresses (of sending and receiving systems) and the byte count for the data frame. The trailer con-

TRANSMISSION FRAMES:

frame	= {control frame, data frame}	
control frame	= {enq,pre,ack,nak,req,nm,eot}	
data frame	= header + packet + trailer	
data frame type	= {msg, lstmsg}	
header	= data frame type + sending system id + receiving system id	
packet	= 10 bytes of ASCII data	
trailer	= checksum (2 bytes ASCII)	
enq	= 'ENQ*'	- handshake inquiry
pre	= 'PRE*'	- handshake acknowledgement
ack	= 'ACK*'	- positive acknowledgement
nak	= 'NAK*'	- negative acknowledgement
req	= 'REQ*'	- request for message
nm	= 'NM'	- no message indicator frame
msg	= 'MS'	- message frame header
lstmsg	= 'LM'	- last message frame header

FIGURE 2.5

tains the checksum.

The message is sent from one system to another in the form of successive data frames. The type of the last data frame is different from the rest of the data frame in order to indicate the end of transmission.

Fig. 2.5 illustrates various frames and their structures.

2.5 ERROR DETECTION:

Errors are caused during the transmissions for a variety of reasons. The data that is sent from one machine to another needs to be checked for transmission errors when it is received at the receiving end.

Simple error detection that is implemented at the physical level is parity. The data sent over the physical medium is 7 bit data. It has a parity bit appended to it.

The parity checking can detect only an odd number of bit errors in one character : it will be able to detect errors that involve one, three, five or seven bits, but not the ones that involve two, four or six bits. Therefore, there needs to be a way of detecting multiple errors which might go undetected in parity checking.

Other mechanisms often employed for error detection are 'Checksum' and 'Polynomial Code' (also known as Cyclic Redundancy Code or CRC). In the checksum mechanism, a checksum or some similar function (like a hash algorithm) is used to compute a number based on

the contents of the data block being transmitted. This number is transmitted alongwith the data block, and verified with the number computed at the receiving end to ensure that the data received does not contain any errors.

Polynomial Codes are based upon treating bit strings as representations of polynomials with coefficients 0 and 1. The sender and the receiver must agree upon the a generator polynomial, G(x), in advance. To compute the checksum for some message with m bits, corresponding to the polynomial M(x), the message must be longer than the polynomial. The basic idea is to append the checksum at the end of the message in such a way that the polynomial represented by the checksummed message is divisible by G(x). When the receiver gets the checksummed message, he attempts to divide it by G(x). If he gets a remainder, it means that there was an error during transmission.

CRC is a very effective method of error detection. However, it imposes significant overhead on the system and slows it down. Since the packets in this design are fairly small (10 bytes), it was decided to use a checksum to detect transmission errors.

For each data packet sent over the communication lines, the checksum of this packet is calculated by adding the integer values of all ASCII characters in the packet, and sent in the trailer that follows the packet. (As explained in Chapter 3, the characters contained within a data packet are in printable ASCII range only.) Upon receipt of the data packet, the receiver computes the check-

sum for the data packet. If the computed checksum matches the checksum received in the trailer, the receiver concludes that there were no errors in the transmission process. If the computed checksum does not match the received checksum, then the receiver requests the sender to retransmit the frame. For the sake of simplicity, the checksum is computed only for the data packet; the header is not included in the checksum. Also, the checksum is computed only for the data frames. Control frames are not checksumed. The 'checksum' algorithm computes the checksum by adding the ASCII equivalents of the characters being transmitted. To avoid the possibility of sending a numeric checksum that might be interpreted by the port drivers as control characters, it needs to be ensured that the checksum is encoded into printable ASCII range. This is done as explained below:

Checksum byte 1 = Integer (checksum/Size of the packet)

Checksum byte 2 = Remainder(checksum/Size of the packet)

Since each byte in the data packet is in the printable ASCII range as discussed in Chapter 3, the first byte will also be in the printable ASCII range. The remainder, which is between 0 and 9, is encoded into its ASCII equivalent by adding equivalent of ASCII 0 to it.

2.6 EFFICIENCY OF ERROR DETECTION ALGORITHM:

The efficiency of an error detection algorithm is determined by how well the algorithm detects the errors in transmission of data.

Higher is the probability of error detection, more efficient is the error detection algorithm.

Errors during transmission normally involve flipping of bits in the data stream being transmitted. Suppose the probability that 'n' bits flip is 'P'. (If the probability of one bit flipping is 'P1', and the probability of 'm' bits flipping is 'Pm', there may or may not exist a relationship between P1 and Pm.) Following is an analysis of the probability (as a percentage of P) that the checksum algorithm will detect this 'n' bit error; for various values of 'n'.

It is assumed that the packet contains 10 data bytes, each byte is made up of 7 bits. (Eighth bit may be used by the physical layer for checksum or some low level protocol.)

CASE 1: n=1

If one bit flips, the value of the byte that contains the flipped bit changes. This change does get reflected in the checksum, and the error will be detected with 100% probability.

CASE 2: n=2

Out of the 70 data bits in the packet, 2 bits can flip in $\binom{70}{2}$ different ways. [The notation $\binom{m}{n}$ represents the number of combinations of n items randomly selected out of a pool of m items at a time.]

Most of these combinations will lead to a change in the checksum;

however, a few will not change the checksum. If a bit in position 5 in one byte flips, the checksum will not change only if bit 5 in any other byte flips in the opposite direction. There are total 10 bits in position 5, and two of these bits can be selected in $\binom{10}{2}$ different ways. Out of the 4 combinations of two bits flipping, only 2 combinations that involve flipping in opposite directions will lead to balancing of the checksum. Therefore, $\frac{1}{2}\binom{10}{2}$ occurrences of flipping bits in position 5 will lead to no change in the checksum. Multiplying this by all 7 possible bit position, the total number of combinations where two bit flippings will not lead to a change in the checksum, are . Therefore, the probability that the checksum will not change is

$$\frac{\frac{7}{2}\binom{10}{2}}{\binom{70}{2}} = 0.065$$

Therefore, the probability that a two bit error will be detected by the checksum algorithm is 1.00 - 0.065 = 0.935 or 93.5%.

CASE 3: n=3

Out of the 70 data bits in the packet, 3 bits can flip in $\binom{70}{3}$ different ways.

If a bit in position 7 in one byte flips from 0 to 1, it can be balanced by flipping of two other bits from 1 to 0 in position 6. These two bits can be selected out of 10 position 6 bits in $\binom{10}{2}$ different ways. Only a fourth of these combinations (combinations 011 and 100 out of the eight possible combinations) will not af-

fect the checksum. If this computation is repeated from bit position 7 to bit position 2, the total number of combinations that do not result in a change in the checksum are $\frac{6}{4}\binom{10}{2}$. Therefore, the probability that the checksum will not change is:

$$\frac{\frac{6}{4}\binom{10}{2}}{\binom{70}{3}} = 0.0012$$

Therefore, the probability that a three bit error will be detected by the checksum algorithm is 1.00 - 0.0012 = 0.9988 or 99.88%.

CASE 4: n=4

The 4-bit error probability can be viewed as a sum of three terms:

- Probability of a 0 bit and a 4 bits error
- Probability of a 1 bit and a 3 bits error
- Probability of a 2 bit and a 2 bits error

The case with 1 bit error will go detected with 100% probability, and hence can be excluded. The case with 4 bit error without affecting the checksum will be very rare, and can be neglected for the sake of simplicity. The probability for the third case (with two 2 bits errors) can be expressed as a product of two terms: first term being the two bit error detection probability in 70 bytes, and the second term being the two bit error detection probability in the remaining 68 bytes. If the later term is approximated to be equal to the earlier, then the probability that a four bit error will go undetected is 0.065 * 0.065 = 0.0043. Therefore, the probability that a four bit error will be detected by

the checksum algorithm is 1.00 - 0.0043 = 0.9957 or 99.57%.

It is obvious that as the value of n increases, the combinations that will not lead to a change in the checksum become more and more scarce. Therefore, the efficiency of the algorithm increases with n.

The above analysis holds good for the checksum error detection alone. If the checksum error detection is combined with the parity error detection, the error detection becomes much more reliable. Parity will detect any odd number of bit flippings with a 100% probability. With the even number of bit flippings, the number of combinations that lead to no change in the checksum and no change in the parity will be even fewer, and the efficiency of the protocol will increase.

One final note on the error detection - the bit flipping errors have a tendency of occurring in bursts. In other words, if a bit in position m in a data packet flips, there is a very high probability that a bit in position m+1 will also flip. More localized the bit flippings are, lesser is the probability that the checksum will not be affected, and therefore, the probability of error detection is high.

3.1 TOPOLOGY:

The details of the five machines are presented in Appendix A.

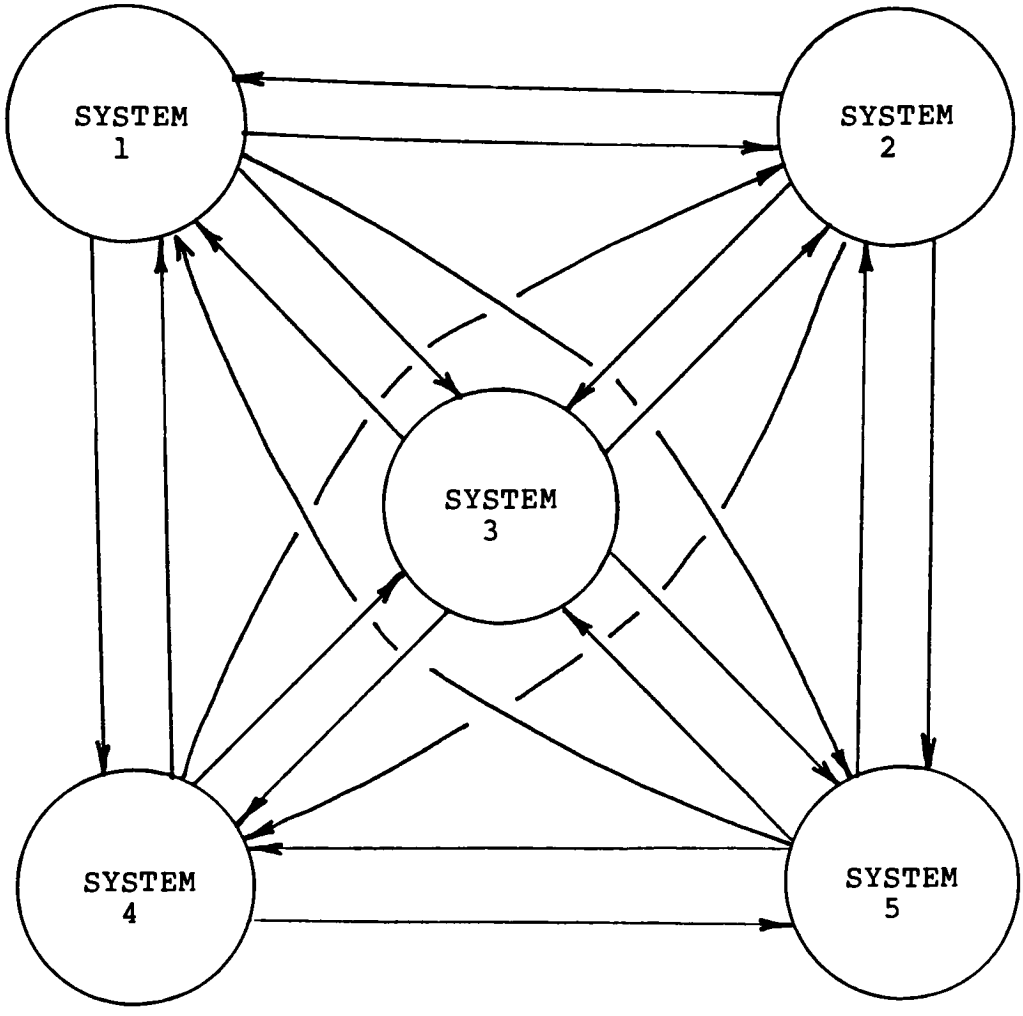
The following two alternatives were considered for establishing communications between processors:

1. All the systems can directly communicate with each other over a dedicated physical medium between each pair, using a standard data-link protocol. (Fig. 3.1)

2. One system can serve as a 'Message Server'. This system will act as a 'Mail-Box' facility for all pairs of systems that want to communicate. The sending system will pass on the message to the message server with the address of the system for whom the message is intended. The message server will store this message, and deliver it to the system for whom it is intended, when that system asks for that message. (Fig. 3.2)

Alternative 2 has been selected for the implementation of this package due to the following considerations:

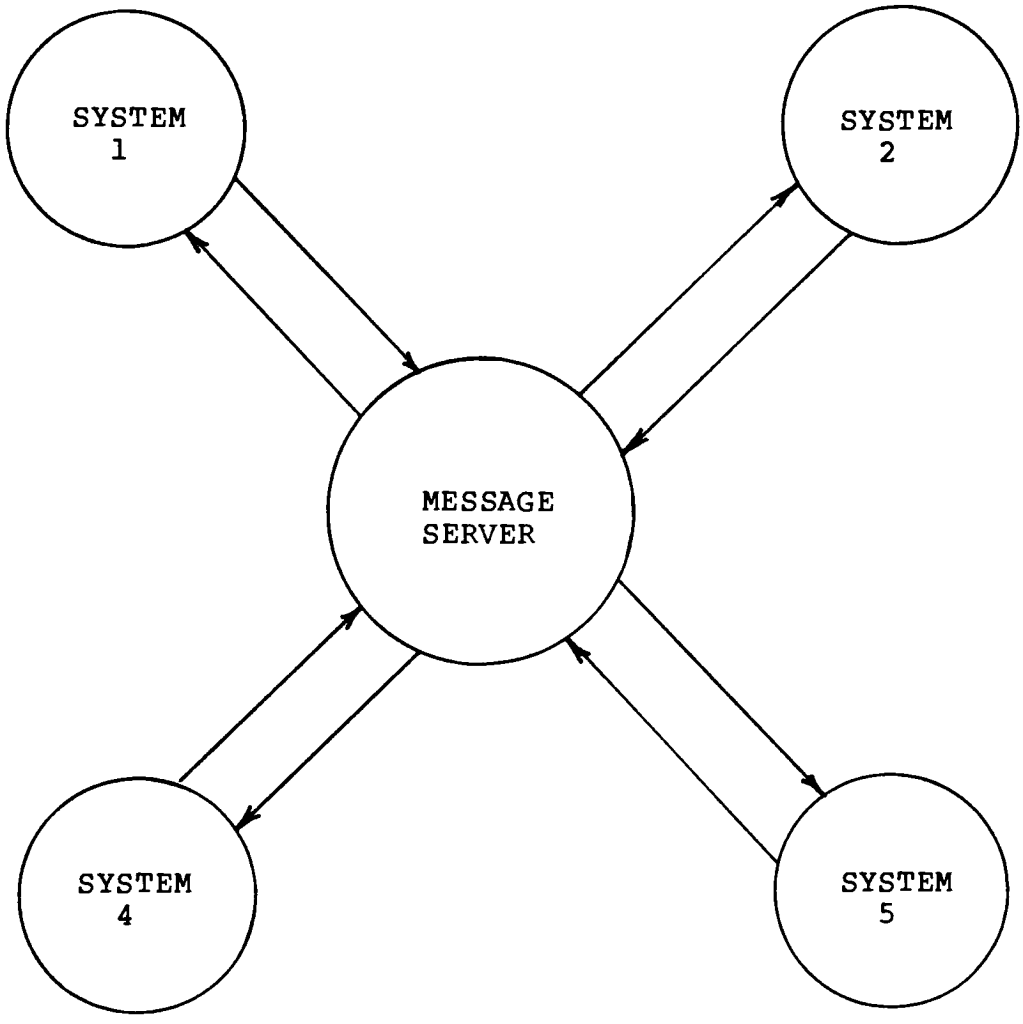
1. Alternative 1 requires that all the systems need to be physically connected to all other systems via RS-232C lines. Most of the systems do have RS-232C ports; howev-



Network Topology Alternative 1

Direct communication between individual systems

FIGURE 3.1



Network Topology Alternative 2

Message Server

FIGURE 3.2

er, with this scheme each system will require 4 different RS-232C ports in order to communicate with 4 different systems. Moreover, with the addition of another system to the network, an additional port will be required on all the systems. Alternative 2 will require only the Message Server system to have enough ports to be able to communicate with all the other systems; all the satellite systems need only one port, regardless of the number of systems in the network.

2. With alternative 2, a 'pseudo' network layer can be introduced to 'route' the messages to their destinations via the message server. If the network expands in future, then this layer can be expanded to a full-service Network Layer that can take care of static and dynamic routing, flow control, congestion control, crash recovery, etc. As illustrated in Fig.3.3, this can be achieved by running the Message Server software on more than one systems to enable each one of them to communicate with a subset of the network. The messages can be stored by the intermediate message servers, and routed to the destination system through other message servers in the network.

3. With alternative 1, messages cannot be transferred until the sending system and the receiving system are in synchronization with each other. With alternative 2, the sending system can send a message even if the receiving

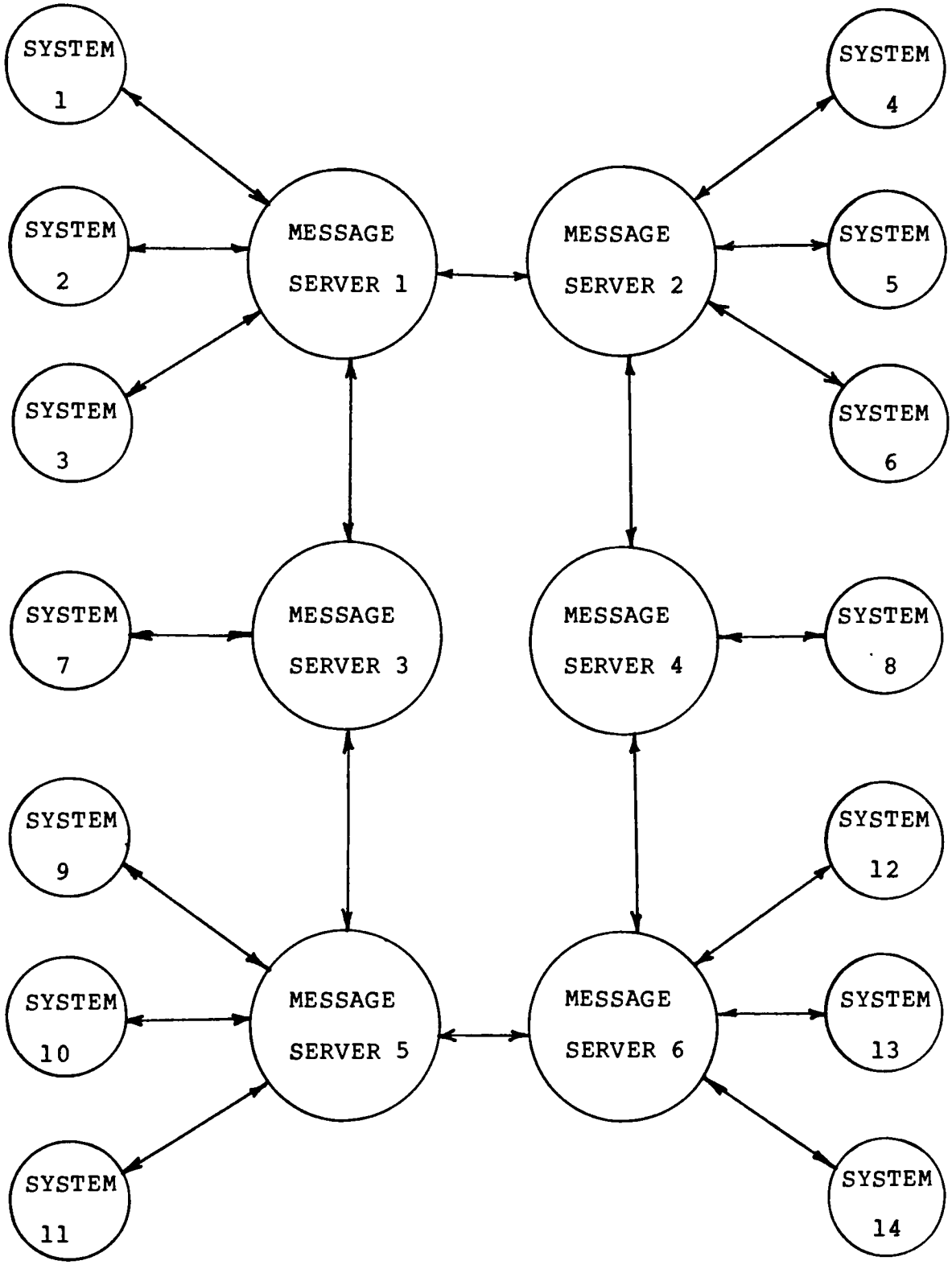


FIGURE 3.3: NETWORK OF MESSAGE SERVERS

system is not up and running at that time. The intended receiver can always receive that message at a later point in time.

Based on the above considerations, a 'Star' network shown in Figure 3.2 was selected for the implementation of this communication package.

3.2 USER INTERFACE:

The tasks running on the satellite systems will be able to send a message to any other satellite system connected to the Message Server by executing the following subroutine call:

```
call send (system_id, buffer_address, nchar, status)

where   system_id      : id of the receiving system
        buffer_address: address of message buffer
        nchar          : number of characters to be sent
        status         : status code returned

Possible values of 'status' are
                          transmission successful
                          message server not up
                          errors in transmission
```

The list of valid destination systems is maintained by the message server in the form of a table. Systems can be added to or removed from the network by adjusting the entries in this table. The data

received by the message server is received and validated by the Message Server and is saved in the mailbox. The receiving system can request this data by executing the following subroutine call:

```
call receive (buffer_address, status)

where   buffer_address: address of receiver buffer
        status        : status code returned

Possible values of 'status' are
                        transmission successful
                        msg server not up
                        errors in transmission
                        no messages waiting
```

The receive call fetches the first available message for the requesting system. It returns an appropriate status code if no messages were available for the requesting system.

The receiving system gets the first message regardless of the system from which the message was received. In future, the 'RECEIVE' call can be expanded to specify the system id from which the message is expected. (See Chapter 5.)

If the sending and receiving tasks are running simultaneously on the two satellite systems, then the message server will deliver the message to the receiving system immediately after the message is received. The receiving system can be in a loop to receive messages. It can keep receiving messages until a pre-designated del-

imiter is received. Since the communication delays are not significant, the transmission of data will be almost instantaneous.

An inter-system file-copy application based on these routines is discussed in Chapter 5.

3.3 MESSAGE SERVER:

The Message Server runs on the system that has physical connections to all satellite systems. The message server is in one of the two modes: 'Poll' mode or 'Process' mode. It can be polling the satellite ports to check for the arrival of a control frame from one of the satellites. This is referred to as 'poll' mode. After the detection of a control frame at a particular satellite port, a handshaking signal is sent to that satellite, and the Message Server enters 'process' mode for that port. In the 'process' mode, the message server processes the communications request to that port alone, and does not keep polling other ports. The input received at any of the other ports is handled when the Message server gets out of the 'process' mode back into the 'poll' mode.

While the message server processes one port in 'Process' mode, the control frames received at the other ports which are in 'Poll' mode are buffered. These frames are processed after the servicing of the port already in 'process' mode is complete.

The design of the Message Server is presented in Fig. 3.4

Ideally, the message server should be designed to poll all the

PSEUDO CODE FOR THE MESSAGE SERVER

```
start polling all the ports

do { for each port
        do {  check for the frame arrival
              if a frame has arrived
                    if handshake request
                              {  send handshake response
                                 switch to process mode
                                 process the port
                                 switch to poll mode
                              }
                    else
                              send a negative response
                    endif
               endif
        }

} forever
```

FIGURE 3.4

PSEUDO CODE FOR IDEAL MESSAGE SERVER

```
do {  set timer and wait for event
      if event = frame arrival
             { if mode = poll
                  { if frame type = handshake request
                       { send a handshake response
                         mode = prpcess
                       }
                    else (if frame type <> handshake request)
                         send a negative response
                    endif
                  }
               else (if mode = process)
                       process the frame
               endif
             }
      endif
} forever
```

FIGURE 3.5

ports at all times and handle communications over various ports at the same time. The message server can be in 'poll' mode for some ports and in 'process' mode for other ports. This way, if a communications session with a particular port takes a long time, the other ports will not have to wait for being serviced until the abovementioned communications session is complete. However, this approach was very difficult to implement on the available hardware due to the lack of various fundamental services like timers and interrupts; hence a simplified approach described in Fig. 3.4 had to be selected.

The 'ideal' design of the Message Server is presented in Fig. 3.5

3.3.1 Subroutine 'POLL'

This subroutine starts poll on the specified port. This is a system-dependent routine, since the mechanism to start the polling is different on different systems. The details of this routine are discussed in Chapter 4.

3.3.2 Subroutine 'PROCESS'

As discussed earlier, this subroutine processes the communications at a particular port when the Message Server enters into 'Process' mode. The satellite, after establishing the communication with the Message Server, may either want to 'send' a message to the message server for putting into the mailbox for the destination system; or it may want to request a message that may have been sent to it by

some other satellite system.

The pseudo code for the subroutine PROCESS is presented in Fig. 3.6

3.3.3 MAILBOX

The mailbox is a two dimensional array in which the messages are stored. For each system, NEXT_IN and NEXT_OUT pointers are maintained to facilitate the insertions and removal of the messages.

3.3.4 Subroutine 'SEND'

The subroutine 'SEND' sends a message to the requesting system. This subroutine breaks the message down into packets of fixed length as described in Chapter 2, builds a frame by adding a header and a trailer to the packet, and sends one frame at a time. Each packet is accompanied by its checksum to detect the transmission errors. The receiving system receives the frames; separates them into data packets, header and trailer; computes the checksum of the data packet, and checks the validity of the packet; and sends a positive or negative acknowledgement to the Message Server. The Message Server transmits the next frame if it receives positive acknowledgement for the previous frame; or re-transmits the previous frame if it receives negative acknowledgement. The transmission attempt is aborted if a frame fails to reach the destination system within 6 attempts.

The pseudo code for subroutine 'SEND' is shown in Fig. 3.7

PSEUDO CODE FOR PROCESS

```
if frame_type = request
        if message exists for this satellite
                get the message from the mailbox
                send the message
                mode = poll
        else
                send 'no_message' frame
                mode = poll
        endif
else if frame_type = message
        receive the message
        post it in the mailbox
        mode = poll
else (if any other type of frame)
        indicate error
        mode = poll
endif
```

FIGURE 3.6

PSEUDO CODE FOR SEND

```
repeat {
        get a packet
        build frame for transmission
        transmit the frame
        wait for response

        if response = positive acknowledgement
                do nothing
        else if response <> positive acknowledgement
                repeat {
                        increment tries
                        transmit the frame
                        wait for the response
                } until (response = positive ack or tries > 6)
        endif
        if teansmission unsuccessful
                abort thr attempt
        endif
} until no more packets
```

FIGURE 3.7

PSEUDO CODE FOR RECEIVE

```
repeat { recieve a frame
         analyze the frame
         if checksum is good
                send positive acknowledgement
                append the packet to the message
         else
                send negative acknowledgement
                increment tries
                if tries > 6
                        abort receive
                endif
         endif
} until no more frames
```

FIGURE 3.8

3.3.5 Subroutine 'RECEIVE'

If the Message Server receives a message frame from the satellite system, it passes the control to 'RECEIVE' subroutine. This subroutine receives the frames, computes the checksum and checks the validity of the data and accordingly sends the acknowledgement to the sender. The packets are put together in their proper order to reconstruct the message. The proper ordering of the packets is ensured by the 'stop-and-wait' protocol.

The pseudo code for the subroutine 'RECEIVE' is presented in Fig. 3.8

3.4 SEND AND RECEIVE ROUTINES ON SATELLITES:

The 'SEND' and 'RECEIVE' routines on the satellite systems do not differ much from those in the Message Server. The earlier are called from the user program, whereas the latter are called from the subroutine 'PROCESS' in the Message Server.

The 'SEND' and 'RECEIVE' routines on the satellites establish a communication with the Message Server prior to sending a message or a request. This is done by sending an 'ENQ' control frame and waiting for a response. If a positive response is received from the Message Server, then the satellite programs proceed with actual data transmission. Error status is returned to the requester if an attempt to establish communications with the message server fails.

The pseudo code for 'ideal' SEND and RECEIVE routines is presented in Fig. 3.9 and 3.10 respectively. As in the case of the Message Server, the satellite programs also should make the use of timers and interrupts to perform I/O to these facilities, the I/O has to be implemented in simpler manner.

PSEUDO CODE FOR AN IDEAL SEND ROUTINE

```
frame type = enq
send frame
set timer and wait for event
if event = frame arrival
        if frame type <> pre
                return failure
        else
                send first packet
        endif
else if event = timeout
        frmae type = eot
        send frame
        return failure
endif

do { set timer and wait for an event
     if event = frame arrival
        read the frame
        if frame type = ack
                send next packet
        else if frame type = nak
                send packet again
        endif
     else if event = timeout
        frame type = eot
        send frame
        return failure
     endif
} forever
```

FIGURE 3.9

PSEUDO CODE FOR AN IDEAL RECEIVE ROUTINE

```
frame type = enq
send the frame
send timer and wait for response
if event = framearrival
        if frame type <> pre
                return failure
        else
                frame typr = req
                send frame
        endif
else if event = timeout
        return failure
endif

do {  set timer and wait for an event
      if event = framearrival
          read frame
          if frame type = nomsg
                return nomsg
          else if frame type = message
                process packet
          else if frame type = eot
                cleanup
                return failure
          endif
      else if event = timeout
          cleanup
          return failure
      endif
} forever
```

FIGURE 3.10

CHAPTER 4 IMPLEMENTATION

4.1 SELECTION OF MESSAGE SERVER

The machine selected to run the message server software was the Perkin Elmer-3220, based on the following considerations.

4.1.1 Number of Ports: Of all the available machines, the Perkin Elmer has the maximum number of RS-232C ports (8). The Tektronix has 3 RS-232C ports, and the DEC Professional-350 has 4. However, the ports on the Pro-350 cannot be used interchangeably, since they do not all have the same characteristics. Therefore, in order to facilitate communications between the maximum number of systems, the Perkin-Elmer is preferred as the message server machine.

4.1.2 Polling: As described in Chapter 3, the ideal way for a program to perform input/output from/to the ports is to queue an I/O request to the port, and continue with the execution of the program. Whenever the input/output is completed, or when the timer fires, the main program gets interrupted and processes the input.

The Tektronix machine has no way of queuing such I/O requests at the ports, and hence could not be considered as a possible choice for the message server.

The DEC Professional-350 does have the most elaborate commands to queue I/O requests to the ports with or without the timer options. These commands are:

```
CALL WTQIO(fnc,lun,efn,pri,isb,prl,ids)
```

where

fnc: I/O function code

lun: logical unit number

efn: Event flag to be set after I/O is complete

pri: Priority

isb: Array to receive status of I/O

prl: Array that contains device parameters

ids: Directive status

The I/O request is terminated after one of the following:

* specified number of characters is processed
* delimiter is processed
* timeout has expired

The delimiter is a specified character that marks the end of data. Normally, the ports are configured to treat the 'carriage return' character as the delimiter. The requesting program can find out the status of the I/O request by checking the event flag associated with the I/O request.

The WTQIO facility of the Professional-350 is sufficient to incorporate the ideal Message Server algorithm indicated in Fig. 3.5. The implementation of 'POLL' routine using the WTQIO call is illustrated in Fig. 4.1. However, the WTQIO command works this way only on one of the four ports on that machine, the printer port. The communication port does not accept the delimiter. Therefore, it is necessary to use fixed length frames only. An attempt was

made to use the WTQIO call with the other two ports. It was possible to output the data to these ports, however it was not possible to read input at these ports after an extended effort, so the attempt had to be aborted.

The Perkin Elmer does have a command that allows the user programs to queue an I/O request at any of the ports for a specified number of characters with or without wait. If the request was queued with wait option, the program gets suspended after queuing the request, and continues after the specified number of characters or a carriage return is received. (There is no way to specify the delimiter, CR is the default delimiter. Also, no timeout can be specified.) If the request was queued without wait option, then the requesting program continues the execution after queuing the request; however, it does not get an interrupt or any indication that the I/O request is completed. Therefore, the program has to go back and periodically check whether the I/O request is complete or not.

The syntax of the sysio command is as follows:

```
CALL SYSIO (PBLK,FC,LU,START,NBYTES,RANADD)
```

where

PBLK: I/O control block

FC: I/O function code

LU: Logical Unit Number

START: Buffer address

NBYTES: Number of bytes to be transferred

RANADD: Record # (for random access only)

This command is not entirely sufficient to implement the ideal Message Server algorithm indicated in Fig. 3.5. However, it can at least implement the bare minimum 'POLL' routine to poll all the ports in the 'poll' mode (Chapter 3). The 'POLL' routine using the SYSIO call on the Perkin Elmer is illustrated in Fig. 4.1

4.1.3 Availability: The Perkin-Elmer is heavily utilized by other users in the Lab. However, due to its multi-user operating system, it was possible to do the development on that machine. The availability of Tektronix and Professional-350 is much better. IBM PCs could not be considered as possible choices for the message server due to their unavailability due to their heavy use.

After considering all the above factors, it was decided that the message server will be developed on Professional-350, and then will be ported over to Perkin-Elmer for the final integration.

4.2 ISOLATION OF SYSTEM DEPENDENT ROUTINES:

Since FORTRAN was the only language common on all these machines, it was decided to implement the communications package in FORTRAN to make it easily portable across various machines. However, in order to minimize the changes to the code while porting it from one machine to another, the system-dependent calls were separated in isolated subroutines. These subroutines were developed for each of the systems separately. These subroutines are described below:

4.2.1 PREAD: This subroutine performs a read operation from a specified port. The calling sequence is:

```
CALL PREAD (NPORT,BUFFER,NCHAR)
```

where NPORT: Port Number
BUFFER: Buffer to read in
NCHAR: Number of chars to be read

Normally, the number of characters is not used, since the number of characters expected at the port is not known in advance. The specified delimiter (normally a 'carriage return' character) terminates the I/O. The delimiter is specified in the system level I/O calls described earlier.

4.2.2 PWRITE: This subroutine performs a write operation to a specified port. The calling sequence is:

```
CALL PWRITE (NPORT,BUFFER,NCHAR,STATUS)
```

where NPORT: Port Number
BUFFER: Buffer to write from
NCHAR: Number of characters to be written
STATUS: Success/Failure flag returned

It may be necessary to append a 'CR' to the string if the output is performed by using a system call that outputs a specified number of characters.

4.2.3 POLL: This subroutine is called to start a poll on the specified port. The calling sequence is:

CALL POLL (NPORT)

Where NPORT: Port Number

This routine is normally called during the initialization phase of the Message Server to start the poll on all the ports. It may also be called to restart the poll on a particular port after handling of the communications at that port is complete. This subroutine will need to be ported only if the message server software is to be ported onto another machine.

4.2.4 WAIT: This routine is called to implement a wait in the program. The calling sequence is:

CALL WAIT(NT)

where NT: wait in seconds

Wait can be implemented by a system call or a DO loop if there is no wait call available on the system.

4.3. SATELLITE PROGRAMS:

The pseudo code for the send and receive routines on the satellite is shown in figure 3.9 and 3.10. The satellite machines attempt to send a frame to the message server and wait for the response by setting a timer. If the response fails to arrive before the timer fires, then the transmission is re-initiated.

In practice, the facility of setting up the timers is not available on all the machines. Therefore, there is no way to implement

POLL ROUTINE FOR DEC PROFESSIONAL-350

```
for ports 1 to maxports do
        { queue read request at the port
        }

        do
        { for ports 1 to maxports do
                if event flag(port) {
                        process port
                        queue another read request
                }
        } forever
```

FIGURE 4.1

POLL ROUTINE FOR DEC PERKIN ELMER-3220

```
for ports 1 to maxports do
        { queue read request at the port
        }

        do
        { for ports 1 to maxports do
                if input = 'enq' {
                        process port
                        queue another read request
                }
        } forever
```

FIGURE 4.2

the time-outs. The sender has to send a frame and keep waiting for the response. Similarly, the receiver program running on the satellite has to send the acknowledgement for the previously received frame and keep waiting for the next frame (unless the frame just processed was the last frame). This will not be an acceptable solution in a remote communication environment where various machines send and receive messages independent of each other. However, in a lab environment where a single user has physical access to, and controls, all these machines, this is not such an impractical compromise.

4.4 ENCODING OF DATA:

The data that is sent from one system to another can be numeric data or can be printable ASCII. If the data is printable ASCII, then it is easily read by the communications ports and passed on to the program that requested I/O. But the data that is numeric in nature is often filtered at the device driver at the port level. For example, a data byte that represents character '3' in ASCII notation actually contains a value of decimal 50. Since 50 happens to be within printable ASCII range, it is read by the port device driver and passed on to the requesting program as such. However, if the data byte had a value of numeric '3', it will not be in the printable ASCII range. It will be interpreted as 'control-c' by the port driver. Since 'control-c' represents an interrupt in most of the systems, it was found that sending the data that contained numeric '3' resulted in termination of the receiving program!

Similarly, unexpected consequences were observed when the data sent contained other numeric values not in the printable ASCII range. For example, some of the numeric data bytes were interpreted as commands to change the port characteristics, and therefore, port characteristics were changed arbitrarily.

In order to avoid this problem, it is necessary to ensure that all the data that is being sent from one system to another is in printable ASCII range. This can be achieved by restricting the users to use the send and receive routines for sending only text data, and preventing them from sending numeric data. This poses undue restrictions on the user, and prevents them from sending digital signals in real-time experimentation. Hence, this solution is not acceptable.

Another way of resolving this problem is to include preprocessor routines in the satellite send program. The preprocessor routines will convert the data to be sent into printable ASCII range before sending it over to the other systems. The receive routine, in turn, will convert the data back to its original form upon receipt. This way, the data sent will always contain bytes in printable ASCII range, and will not be filtered at the port level. Therefore, 'encode' and 'decode' routines were introduced in the satellite 'send' and 'receive' programs to do the encoding and decoding of data.

Fig. 4.3 shows how a numeric data byte is converted into printable ASCII range. As can be seen from the figure, the encoded data will

Input data byte

a	b	c	d	e	f	g	h

High Nibble Low Nibble

Will be encoded into following two bytes

0	0	1	0	a	b	c	d

High Nibble Low Nibble

and

0	0	1	0	e	f	g	h

High Nibble Low Nibble

FIGURE 4.3 : ENCODING OF DATA

always be in the printable ASCII range, that extends from binary 0010 0000 to 0111 1110.

4.5 PROBLEMS DURING IMPLEMENTATION

It may be helpful to summarize various implementation related problems and how they were resolved.

4.5.1 Echo: Many port drivers echo the characters as they are read at the port. It was observed that Perkin Elmer and DEC Professional-350 ports echoed the characters. In Professional-350, an option to turn off the echo on a specified port was available. But no such option was available with the Perkin Elmer. Therefore, when a Perkin Elmer port read an 'enq' sent to it by another system, it sent the same character sequence back to the satellite as an echo. The satellite did not expect an echo, therefore it read that sequence and treated it as a response from the message server for its 'enq'. Since the response was not as expected, the communication could not be established.

The problem was dealt with by simply introducing a 'read' statement after every write in the routine 'PWRITE' in the satellite programs. This way, the echo was trapped before the response from the message server could be read. If the message server software is to be run on a machine that does not echo the characters read at the port, then it needs to be modified to generate artificial echo, to be consistent with the systems that generate the echo.

4.5.2 Mismatching Speeds: Since various machines execute instruc-

tions at different speeds, it was sometimes not possible to get them in synchronization with each other. For example, system A sends a frame to system B over a port P, and waits for the response by executing a 'PREAD' call on that port. If the system B is very fast compared to system A, the response from system B will be delivered to port P on system A before the execution of the 'PREAD' call on the system A was complete. In this case, the response sent to the system A from the system B will be lost, if there is no buffering of the characters at the port level. The only way to resolve this problem is to introduce delays in the 'PWRITE' routines on all the systems to allow other systems to complete the execution of the corresponding 'PREAD' calls.

4.5.3 Control Characters: As mentioned earlier, the numeric data was sometimes interpreted by the port drivers as control characters, and unexpected results followed. This problem was resolved by encoding the data into printable ASCII range as described in the previous article.

5.1 APPLICATIONS:

Simple test programs to send data from one system to another were initially used to ensure that the basic logic of this communications package was sound. The initial tests were conducted by connecting the systems to a terminal instead of another system, and manually simulating the response of another system. The encoding and decoding of data was tested by matching the encoded data packet with the encoding algorithm, and thereafter verifying that the decoded packet is same as the original packet. Tests were conducted to verify that numeric data can be sent from one system to another.

The package was rather slow due to the delays introduced in the I/O routines, and the use of FORTRAN to do the I/O. It took a few seconds to send a message that was 80 characters long. The speed of the package can be improved considerably by implementing some of the enhancements mentioned later in this chapter.

An inter-system file copy application can be developed by using this package. The sending system can be in a loop to send messages until there are no more records to be sent. The receiving system can receive these messages in a loop and write them out to a file, until it receives the 'end of file' indicator. The pseudo-codes of the file copy programs running on both the systems

are illustrated in Fig. 5.1

Using this package, systems can exchange data between each other in 'real-time' environment under program control. One such application, illustrated in fig. 5.2, is the graphic representation of a numerically controlled lathe in the Industrial Engineering Laboratory on Tektronix-4113. The lathe can be controlled by the DEC Professional-350 via the digital I/O lines. The Professional-350 can send the status of these digital I/O lines to Tektronix-4113 using the communications package described in this thesis. The lathe then can be monitored and controlled from the graphic display on the Tektronix machine.

5.2 ENHANCEMENTS:

There are numerous other enhancements that could be done to the existing message server to make it more efficient and easy to use. They are as follows:

- System-dependent PREAD and PWRITE routines could be written in assembly language to make the I/O faster.
- The delays in the I/O routines can be 'tuned' to reduce the communication time considerably.
- The services of the 'Session Layer' to convert the data from the format of the sending system to that of the receiving system could be provided as a part of this communications package.

PSEUDO-CODE FOR SENDER PROGRAM:

```
open the file
repeat { read a record
         send the record
} until no more records
send the end-of-file indicator
```

PSEUDO CODE FOR RECEIVER PROGRAM:

```
open a file
repeat { receive a message
         if message contains data record
                append the data record to the file
         else if the no message indicator
                do nothing
         endif
} until message = end-of-file indicator
```

FIGURE 5.1 : INTER SYSTEM FILE COPY UTILITY

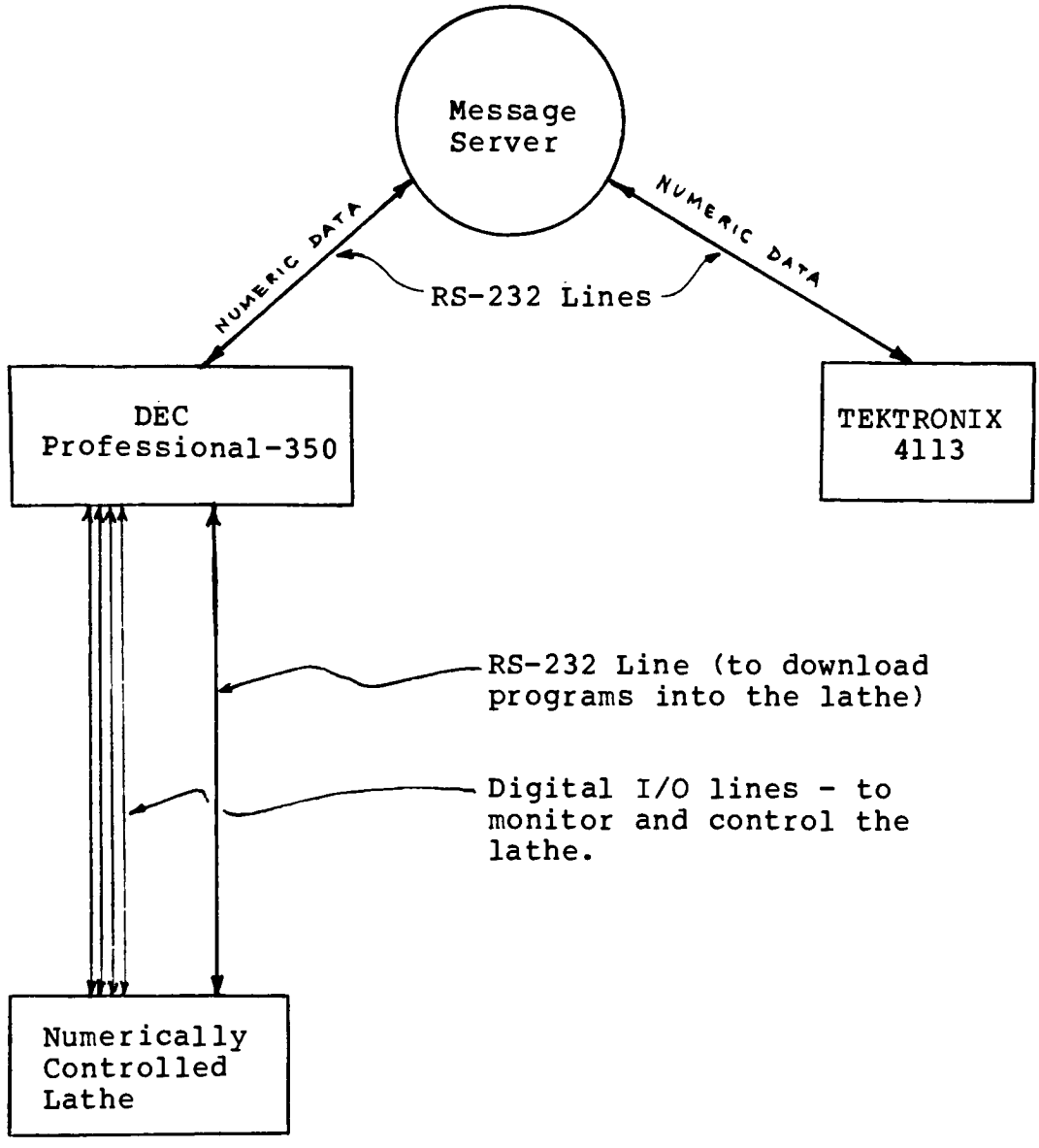


FIGURE 5.2 - USE OF THE COMMUNICATION PACKAGE FOR EXCHANGE OF DATA TO CONTROL A MACHINE IN REAL-TIME.

- RECEIVE routine could be enhanced to specify the size of the buffer. If the message received is larger that the buffer, then an error would be returned.

- A function could be provided to check whether messages exist for the calling system.

- A system could be able to request to receive a message from another specified system. Presently, the RECEIVE routine returns the first available message in the mailbox for the requesting system regardless of the sending system. The message server could be modified to maintain the mailbox in an order that is sorted by the sending system ID. This will enable the message server to deliver a message to the requesting system from a specified system only.

- As described in Chapter 4, the data bytes in the data frame are encoded in order to make sure that the data transmitted is within printable ASCII range. The encoding algorithm implemented encodes one data byte into two characters, thus doubling the data flow. The characters produced by the encoding algorithm are restricted to a very narrow range (32 to 47) of the printable ASCII character set (32 to 126). The encoding algorithm could be modified to encode more than one byte into two characters by making the use of the full printable ASCII range. For example, two data bytes could be encoded

into three characters, thus reducing the amount of data flow.

5.3 EXPANSION OF THE NETWORK

The communication package as presented in this writeup was designed to meet the present needs for the communication between the systems. This package will need enhancements if the network grows in size, and/or more functionality is needed. This section describes some of these enhancements

5.3.1 ROUTING: With a larger network, it may not be possible to have a single message server to be physically connected to each system in the network. Therefore, there will be a need for multiple message servers connected to each other as illustrated in Fig. 3.3. Each message server will be connected to a group of local systems and will communicate with all other systems through a network of message servers. The topology of the network will be known to all the message servers. If message server 1 receives a message from system 1 for system n, then the message server 1 will find out the path to message server m that is directly connected to system n, and route this message to the message server m through intermediate message servers. Message server m will store this message until system n asks for the message.

In order to facilitate the routing, both the data and control frames need to contain an address of the sending and receiving system.

5.3.2 CRASH/RECOVERY: The routing tables maintained at each message server will have to be kept up-to-date and consistent with each other. Crash or recovery of any message server needs to be made known to all the other message servers in the network as early as possible.

5.3.3 PROTOCOL: If the network expands, the existing stop-and-wait protocol may result in very inefficient utilization of the network resources and consequently poor throughput. This is because the transmission delays could be significant. It may become necessary to replace the existing simple protocol by a sliding window protocol which will allow overlap between the transmission of various messages, thus increasing the throughput. This may need a considerable restructuring of the message server logic.

APPENDIX 1 DETAILS OF SYSTEMS

---

1. NAME : Perkin Elmer-3220
   MANUFACTURER : Perkin Elmer, INC.
   PROCESOR : Proprietary
   OPERATING SYSTEM: OS-32
   MEMORY : 768 K
   STORAGE : 10 MB cartridge disks, 2 Nos.
   COMM. PORTS : RS-232C ports - 8 Nos.
   Additional digital I/O ports

2. NAME : Tektronix-4113A
   MANUFACTURER : Tektronix, INC.
   PROCESOR : Intel-8086
   OPERATING SYSTEM: CPM/86
   MEMORY : 512 K
   STORAGE : 400 MB Floppy disks (8 inch) - 2 Nos.
   COMM. PORTS : RS-232C ports - 3 Nos.
   Additional host port

3. NAME : Professional-350
   MANUFACTURER : Digital Equipment Corporation, INC.
   PROCESOR : Proprietary (PDP-11 s1 processor)
   OPERATING SYSTEM: RSX-11M+
   MEMORY : 512 K
   STORAGE : 400 KB Floppy disks (5 1/4 inch) - 2 Nos.
   COMM. PORTS : RS-232C ports - 4 Nos.

additional digital I/O ports.

4. NAME : IBM - PC-XT
   MANUFACTURER : IBM, Inc.
   PROCESOR : Intel-8086
   OPERATING SYSTEM: PC-DOS
   MEMORY : 512 K
   STORAGE : 10 MB hard disk
   360 KB floppy disk (5 1/4 inch)
   COMM. PORTS : RS-232C port - 1 No.

5. NAME : IBM PC-AT
   MANUFACTURER : IBM, Inc.
   PROCESOR : Intel-80286 (Adv. Tachnology)
   OPERATING SYSTEM: PC-DOS
   MEMORY : 512 KB
   STORAGE : 10 MB hard disk
   360 KB floppy disk (5 1/4 inch)
   COMM. PORTS : RS-232C port - 1 No.

1. MESSAGE SERVER

```
      PROGRAM MYMSG
C     THIS IS THE MESSAGE SERVER PROGRAM THAT SERVICES ALL THE PORTS
C     OF THE MESSAGE SERVER UNIT.
C     NPORTS SPECIFIES THE NUMBER OF PORTS
      INTEGER STRUE,SFALSE
C     CONSTANTS TO INDICATE STRUE/SFALSE FLAGS
      INTEGER OK
C     STATUS FLAG CONSTANT
      DIMENSION NIN(5),NOUT(5)
C     IN & OUT POINTER ARRAYS FOR MAILBOX
      CHARACTER*80 STRING
      CHARACTER*80 MAIL(5,10)
C     ARRAY THAT STORES MESAGES
      DIMENSION LENGTH(5,10)
C     ARRAY THAT STORES LENGTHS OF MESSAGES
C     MAIL(I,J) INDICATES MESSAGE NUMBER J FOR SYSTEM I
C     LENGTH(I,J) INDICATES LENGTH OF MESSAGE MAIL(I,J)
      CHARACTER*8 PROMPT(5)
      INTEGER IPORT(10),IFLAG(10)
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT
      CHARACTER*4 REQ
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
      COMMON/BLK4/MAIL
      COMMON/BLK4A/LENGTH
      COMMON/BLK5/NIN,NOUT
      COMMON/BLK6/MAXMSG,MXPRTS,STRUE,SFALSE
      COMMON/BLK7/OK
      COMMON/BLK8/IPORT,IFLAG,ISIZE
      COMMON/BLK9/PROMPT
C     INITIALIZATION
      DATA IPORT/9,11,13,15,17,19,0,0,0,0/
      DATA IFLAG/11,12,13,14,15,16,17,18,19,20/
C     WRITE(6,201)
 201  FORMAT('*** ENTERING MSGSRV ***')
      ACK='ACK*'
      NAK='NAK*'
      ENQ='ENQ*'
      PRE='PRE*'
      EOT='EOT*'
      REQ='REQ*'
      NOMSG='NM'
      MSG='MS'
      LSTMSG='LM'
      GOOD='GD'
      STRUE=1
      SFALSE=0
      OK=1
C
      ISIZE=5
C     NUMBER OF CHARACTERS TO BE READ IN PROMPT
      MXPRTS=5
C     MAXIMUM NUMBER OF PORTS.
      MAXMSG=10
C     MAX NUMBER OF MESSAGES STORED FOR ANY SYSTEM
C     IF MXPRTS OR MAXMSG CHANGE, DIMENSIONS OF
C     NIN,NOUT & MAIL ALSO NEED TO BE CHANGED
C
```

```
      NPORTS=2
C     A FAKE CHECK TO EXIT INFINIT LOOP
      IF (NPORTS.EQ.0) GOTO 110
C
C     INITIALIZE NIN AND NOUT FOR ALL PORTS
      STRING='**** WELCOME TO THE MESSAGE SERVER PROGRAM ****'
C     CALL PWRITE(6,STRING,24)
C     CALL PWRITE(12,STRING,20)
      DO 80 I=1,MXPRTS
      NIN(I)=0
      NOUT(I)=0
      DO 85 J=1,MAXMSG
 85   MAIL(I,J)=' '
 80   CONTINUE
C
C     ****************** MAIN LOOP ************************
C
C     THIS LOOP QUEUES READ CALLS TO ALL PORTS
      DO 60 I=1,NPORTS
      CALL POLL(I)
 60   CONTINUE
C
C     NOW A LOOP TO CHECK ALL EVENT FLAGS
 65   DO 70 I=1,NPORTS
C     WRITE(6,206)
 206  FORMAT(' POLLING')
      IFOUND=INDEX(PROMPT(I),ENQ)
      IF (IFOUND.EQ.0) GOTO 70
C     INPUT RECEIVED AT THIS PORT. PROCESS IT
C     WRITE(6,207)
 207  FORMAT(' PROMPT ARRAY CONTAINS FOLLOWING ELEMENTS')
      DO 210 NUM=1,3
C     WRITE(6,208)PROMPT(NUM)
 210  CONTINUE
 208  FORMAT('->',A10,'<->',A10,'<->',A10,'<-')
      CALL PROCES(I)
C     QUEUE ANOTHER READ CALL TO THIS PORT
      CALL POLL(I)
 70   CONTINUE
C     LOOP UP
      GOTO 65
 110  CONTINUE
      END
C
```

```
      SUBROUTINE POLL(I)
      CHARACTER*8 PROMPT(5)
      INTEGER*4 INTINP(10)
      EQUIVALENCE (INTINP,PROMPT)
      INTEGER*4 PBLK(5,5),FC
      INTEGER STATUS(5)
C     INPUT ARRAY FOR ALL PORTS
      INTEGER IPORT(10),IFLAG(10)
      COMMON/BLK8/IPORT,IFLAG,ISIZE
      COMMON/BLK9/PROMPT
      IER=0
      NBYTES=8
      PROMPT(I)='     '
      FC=64
C     64 IS CODE FOR READ WITHOUT WAIT
C     WRITE(6,201)I
 201  FORMAT(' CALLING QIO FOR PORT ', I4)
      CALL SYSIO(PBLK(I,5),FC,IPORT(I),INTINP(2*I-1),NBYTES,0)
      CALL IOERR(PBLK(I,5),STATUS(I))
C     WRITE(6,203) STATUS(I)
 203  FORMAT(' STATUS OF SYSIO IS ',I4)
      RETURN
      END
C
```

```
      SUBROUTINE PWRITE(IUNIT,NTRING,NCOUNT)
C     THIS SUBROUTINE OUTPUTS A STRING 'STRING' OF SIZE 'NCOUNT'
C     PLUS A CARRIAGE RETURN TO PORT IUNIT
      CHARACTER*84 STRING
      INTEGER*4 PBLK(5),FC
      CHARACTER*80 NTRING
      INTEGER*4 BLK(21)
      INTEGER*4 STATUS
      EQUIVALENCE (BLK,STRING)
C     INPUT ARRAY FOR ALL PORTS
      CHARACTER*8 PROMPT(5)
      INTEGER IPORT(10),IFLAG(10)
      COMMON/BLK8/IPORT,IFLAG,ISIZE
      COMMON/BLK9/PROMPT
      FC=40
      STRING=' ' // NTRING
      NCOUNT=NCOUNT+1
C     40 IS FUNCTION CODE FOR WRITE WITH WAIT
      ICR=13
C     CARRIAGE RETURN
      IER=0
C     WRITE(6,205)
 205  FORMAT('*** ENTERING PWRITE ***')
      ITM=3
C     WAIT TIME IN SECONDS
      IUN=2
C     WAIT UNIT(SECONDS)
      IST=0
C     STATUS OF WAIT CALL
C     CALL WAIT(ITM,IUN,IST)
C     WRITE(6,207)IST
 207  FORMAT(' STATUS OF WAIT WAS ',I8)
      NCOUNT=NCOUNT+1
      WRITE(STRING(NCOUNT:NCOUNT),101)ICR
 101  FORMAT(A1)
      NCOUNT=NCOUNT-1
C     WRITE(6,201)IUNIT
 201  FORMAT(' CALLING SYSIO FOR UNIT ', I4)
C     WRITE(6,204) STRING
 204  FORMAT(' THE STRING IS',A80)
C     WRITE(6,211) (BLK(K),K=1,20)
211   FORMAT(' BLK=',20A4)
      CALL SYSIO(PBLK,FC,IUNIT,BLK,NCOUNT,0)
      CALL IOERR(PBLK,STATUS)
C     WRITE(6,203) STATUS
 203  FORMAT(' STATUS OF PWRITE IS ',I10)
C     WRITE(6,206)
 206  FORMAT('*** LEAVING PWRITE ***')
      RETURN
      END
C
```

```
      SUBROUTINE PREAD(IUNIT,NTRING,NCOUNT,NEND)
C     THIS SUBROUTINE OUTPUTS A STRING 'STRING' OF SIZE 'NCOUNT'
C     PLUS A CARRIAGE RETURN TO PORT IUNIT
      CHARACTER*80 STRING
      INTEGER*4 PBLK(5),FC
      CHARACTER*80 NTRING
      INTEGER*4 BLK(20)
      INTEGER STATUS
      EQUIVALENCE (STRING,BLK)
C     INPUT ARRAY FOR ALL PORTS
      NEND=0
      FC=72
C     72 IS FUNCTION CODE FOR READ WITH WAIT
      IER=0
C     WRITE(6,205)
 205  FORMAT('*** ENTERING PREAD ***')
C     WRITE(6,201)IUNIT
 201  FORMAT(' CALLING SYSIO FOR UNIT ', I4)
      CALL SYSIO(PBLK,FC,IUNIT,BLK,NCOUNT,0)
      CALL IOERR(PBLK,STATUS)
C     WRITE(6,203)STATUS
 203  FORMAT(' STATUS OF PREAD IS ',I10)
C     WRITE(6,204) STRING
 204  FORMAT(' THE STRING READ IS',A80)
      NTRING=STRING
      IFOUND=0
      IFOUND=INDEX(NTRING,'ENQ*')
      IF(IFOUND.NE.0)NEND=1
C     WRITE(6,207)NEND
  207 FORMAT(' NEND=',I2)
C     WRITE(6,206)
 206  FORMAT('*** LEAVING PREAD ***')
      RETURN
      END
C
```

```
      SUBROUTINE PROCES(IP)
C     THIS SUBROUTINE PROCESSES THE INPUT 'PROMPT(IP)' FROM A PORT 'IP'
      INTEGER STATUS
      INTEGER STRUE,SFALSE
C     INITIALIZE POLL
      INTEGER OK
C     STATUS FLAG CONSTANT
      CHARACTER*80 STRING,BUFFER
      CHARACTER*8 PROMPT(5)
      INTEGER IPORT(10),IFLAG(10)
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
      COMMON/BLK6/MAXMSG,MXPRTS,STRUE,SFALSE
      COMMON/BLK7/OK
      COMMON/BLK8/IPORT,IFLAG,ISIZE
      COMMON/BLK9/PROMPT
      NPI=IPORT(IP)
      NPO=NPI+1
C     WRITE(6,201)
 201  FORMAT('*** ENTERING PROCESS ***')
      ISYSID=IP
C     WRITE(6,202)IP,IPORT(IP),NPI,NPO
 202  FORMAT(' IP,IPORT,NPI,NPO ARE ',4I8)
C     WRITE (6,203) PROMPT(IP)
 203  FORMAT (' THE PROMPT IS [',A4.']')
C     ERROR IF INPUT IS NOT ENQ
C     WRITE(6,208)ENQ,PROMPT(IP)
 208  FORMAT(' ENQ AND PROMPT ARE --->',A4,'<--- AND --->',A4,'<---')
      IFOUND=0
      IFOUND=INDEX(PROMPT(IP),ENQ)
      IF (IFOUND.EQ.0) GOTO 140
C     ******** HANDLE 'ENQ' *************
 141  STATUS=0
      NEND=0
      STRING=PRE
      CALL PWRITE(NPO,STRING,4)
C     WRITE(6,205)
 205  FORMAT(' JUST WROTE A PRE')
C     GET NEXT INPUT FROM THE SAME PORT
C     WRITE(6,209)NPI
 209  FORMAT(' NPI BEFORE THE READ IS ',I8)
      NCHAR=80
C     NUMBER OF CHARACTERS TO BE READ
      CALL PREAD(NPI,STRING,NCHAR,NEND)
      IF(NEND.EQ.1)GOTO 141
C
      IFOUND=0
      IFOUND=INDEX(STRING,REQ)
      IF (IFOUND.EQ.0) GOTO 120
C     ******** HANDLE 'REQ' *************
C     CHECK WHETHER ANY MESSAGE EXISTS FOR THIS SYSTEM.
      CALL GETMSG(ISYSID,BUFFER,LGT,MFLAG)
      IF (MFLAG.NE.STRUE) GOTO 130
C     ........ MESSAGE EXISTS. SEND IT. ..........
      CALL SEND(ISYSID,BUFFER,LGT,STATUS)
C     REMINDER - PUT THIS CHECK IN AND REMOVE NEXT STATEMENT
      IF (STATUS.EQ.OK)CALL CLEAN(ISYSID)
```

```
      IF(STATUS.EQ.3)GOTO 141
      GOTO 500
C     ........ NO MESSAGE. SEND 'NOMSG'. .........
 130  STRING=NOMSG
      CALL PWRITE(NPO,STRING,2)
      GOTO 500
C
C     ******** MUST BE A MESSAGE. RECEIVE IT. ***********
 120  BUFFER=' '
      CALL RECIVE(ISYSID,STRING,BUFFER,LGT,ISYSTO,STATUS)
C     POST THE MESSAGE IF ONE RECEIVED.
      IF(STATUS.EQ.OK)CALL POST(ISYSTO,BUFFER,LGT,STATUS)
      IF(STATUS.EQ.3)GOTO 141
      GOTO 500
C
C     *********** ILLEGAL MSG **************
C     ILLEGAL INPUT DETECTED, RETURN
 140  STRING=NAK
      CALL PWRITE(NPO,STRING,4)
      GOTO 500
C     ###############  PROCESSING COMPLETE. RETURN #############
 500  CONTINUE
C     WRITE(6,43)
 43   FORMAT('*** LEAVING PROCESS ***')
      RETURN
      END
C
```

```
C     ************************SEND*****************************
      SUBROUTINE SEND(ISYSID,BUFFER,LGT,STATUS)
C     SUBROUTINE TO SEND DATA BUFFER TO SYSTEM 'ISYSID'
      CHARACTER*20 PACKET,FRAME
      CHARACTER*80 BUFFER
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
      INTEGER STATUS
C     WRITE(6,201)
 201  FORMAT('*** ENTERING SEND ***')
      JPAKNO=0
C
 100  CONTINUE
      JPAKNO=JPAKNO+1
      CALL GETPAK(BUFFER,LGT,PACKET,JPAKNO,LAST)
C     WRITE(6,211)LAST
 211  FORMAT('AFTER GETPAK, LAST IS ',I8)
      CALL BLDFRM(PACKET,LAST,FRAME)
C     WRITE(6,212)LAST
 212  FORMAT('AFTER BLDFRM, LAST IS ',I8)
      CALL XMTFRM(FRAME,STATUS)
C     WRITE(6,203)LAST,STATUS
 203  FORMAT('AFTER XMTFRM, LAST,STATUS ARE ', 2I8)
C     ERROR - EXIT
      IF (STATUS.EQ.0)GOTO 500
      IF (STATUS.EQ.3)GOTO 500
      IF (LAST.EQ.0)GOTO 100
C     LAST PACKET - EXIT
 500  CONTINUE
C     WRITE(6,43)
 43   FORMAT('*** LEAVING SEND ***')
      RETURN
      END
C
```

```
C       SUBROUTINE TO BUILD FRAME
        SUBROUTINE BLDFRM(PACKET,LAST,FRAME)
C       INPUT - PACKET
C       OUTPUT - FRAME (MSG TYPE+PACKET+CHECKSUM)
        CHARACTER*20 PACKET
        CHARACTER*20 FRAME
        CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
        CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
        CHARACTER*2 CH
        COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
        COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK3/NPI,NPO
        IFROM=51
        ITO=52
        IC=53
C       WRITE(6,201)
 201    FORMAT('*** ENTERING BLDFRM ***')
        FRAME(1:2)=MSG
        IF(LAST.EQ.1)FRAME(1:2)=LSTMSG
        WRITE(FRAME(3:3),100)IFROM
C       'FROM' SYSTEM
        WRITE(FRAME(4:4),100)ITO
C       'TO' SYSTEM
 100    FORMAT(A1)
        WRITE(FRAME(5:5),100)IC
C       CODE THE COUNT
        FRAME(6:15)=PACKET
        CALL CHKSUM(PACKET,CH)
        FRAME(16:17)=CH
C       WRITE(6,43)
 43     FORMAT('*** LEAVING BLDFRM ***')
        RETURN
        END
C
```

```
C       SUBROUTINE TO GET MESSAGE
        SUBROUTINE GETMSG(ISYSID,BUFFER,LGT,MFLAG)
        INTEGER STRUE,SFALSE
        DIMENSION NIN(5),NOUT(5)
        CHARACTER*80 MAIL(5,10)
        DIMENSION LENGTH(5,10)
C       ARRAY THAT STORES LENGTHS OF MESSAGES
        CHARACTER*80 BUFFER
        COMMON/BLK4/MAIL
        COMMON/BLK4A/LENGTH
        COMMON/BLK5/NIN,NOUT
        COMMON/BLK6/MAXMSG,MXPRTS,STRUE,SFALSE
C       WRITE(6,201)
 201    FORMAT('*** ENTERING GETMSG ***')
        MFLAG=SFALSE
        BUFFER=' '
        IF(NIN(ISYSID).EQ.0)GOTO 110
C       NO MESSAGES PENDING
C       ........ MESSAGES PENDING. GET IT .....
        MFLAG=STRUE
        BUFFER=MAIL(ISYSID,NOUT(ISYSID))
        LGT=LENGTH(ISYSID,NOUT(ISYSID))
 110    CONTINUE
C       WRITE(6,43)
 43     FORMAT('*** LEAVING GETMSG ***')
        RETURN
        END
C
```

```
C     SUBROUTINE TO POST THE MESSAGE IN THE ARRAY OF MESSAGE SERVER
      SUBROUTINE POST(ISYSID,BUFFER,LGT,STATUS)
      CHARACTER*80 BUFFER
      INTEGER STATUS
      INTEGER STRUE,SFALSE
      DIMENSION NIN(5),NOUT(5)
      CHARACTER*80 MAIL(5,10)
      DIMENSION LENGTH(5,10)
C     ARRAY THAT STORES LENGTHS OF MESSAGES
      COMMON/BLK4/MAIL
      COMMON/BLK4A/LENGTH
      COMMON/BLK5/NIN,NOUT
      COMMON/BLK6/MAXMSG,MXPRTS,STRUE,SFALSE
C     WRITE(6,201)
 201  FORMAT('*** ENTERING POST ***')
C     WRITE (6,21)STATUS
 21   FORMAT('FROM POST, THE STATUS IS',I8,' AND THE BUFFER IS')
C     WRITE (6,20) BUFFER
 20   FORMAT('##',A80)
C     WRITE(6,43)
C     IF MAILBOX IS FULL, RETURN ERROR STATUS
      IF((NOUT(ISYSID).EQ.NIN(ISYSID)).AND.(NIN(ISYSID).NE.0))GOTO 100
C     HANDLE SPECIAL CASE WHERE MAIL WAS EMPTY
C     IF NIN=NOUT=0 THEN MAKE NIN=1,NOUT=1
      IF(NIN(ISYSID).NE.0)GOTO 110
      NIN(ISYSID)=1
      NOUT(ISYSID)=1
C     INSERT THE BUFFER IN MAILBOX
 110  MAIL(ISYSID,NIN(ISYSID))=BUFFER
      LENGTH(ISYSID,NIN(ISYSID))=LGT
C     ADJUST THE POINTERS
      NIN(ISYSID)=NIN(ISYSID)+1
      IF(NIN(ISYSID).GT.MAXMSG) NIN(ISYSID)=1
      STATUS=0
      GOTO 120
C     NO ROOM IN MAILBOX. SHOW ERROR STATUS
 100  STATUS=1
 120  CONTINUE
 43   FORMAT('*** LEAVING POST ***')
      WRITE(6,202)
 202  FORMAT('THE FOLLOWING MESSAGES EXIST FOR THE SYSTEMS')
      DO 212 MSYS=1,2
      DO 206 MSGNO=1,10
      WRITE(6,203)MSGNO,LENGTH(MSYS,MSGNO),MAIL(MSYS,MSGNO)
 206  CONTINUE
      WRITE(6,207)NIN(MSYS),NOUT(MSYS)
      WRITE(6,213)
 213  FORMAT(' ..........................................')
 207  FORMAT(' NIN=',I8,' & NOUT=',I8)
      WRITE(6,214)
 214  FORMAT(' __________________________________________')
 212  CONTINUE
 203  FORMAT(I3,'/',I3,'/',A70)
      RETURN
      END
C
```

```
C       SUBROUTINE TO REMOVE MESAGE FROM ARRAY OF MESSAGE SERVER
        SUBROUTINE CLEAN(ISYSID)
        INTEGER STRUE,SFALSE
        DIMENSION NIN(5),NOUT(5)
        CHARACTER*80 MAIL(5,10)
        DIMENSION LENGTH(5,10)
C       ARRAY THAT STORES LENGTHS OF MESSAGES
        COMMON/BLK4/MAIL
        COMMON/BLK4A/LENGTH
        COMMON/BLK5/NIN,NOUT
        COMMON/BLK6/MAXMSG,MXPRTS,STRUE,SFALSE
C       WRITE(6,201)
 201    FORMAT('*** ENTERING CLEAN ***')
        NOUT(ISYSID)=NOUT(ISYSID)+1
        IF(NOUT(ISYSID).GT.MAXMSG)NOUT(ISYSID)=1
C       IF QOE IS EMPTY (NOUT=NIN), RESET BOTH TO 0
        IF(NOUT(ISYSID).NE.NIN(ISYSID))GOTO 110
        NOUT(ISYSID)=0
        NIN(ISYSID)=0
 110    CONTINUE
C       WRITE(6,43)
 43     FORMAT('*** LEAVING CLEAN ***')
        RETURN
        END
C
```

```
C       SUBROUTINE TO RETURN NEXT PACKET
C       INPUT : BUFFER,PACKET NUMBER REQUIRED
C       OUTPUT: PACKET,LAST
C       LAST=0 IF MORE PACKETS, 1 IF LAST PACKET
        SUBROUTINE GETPAK(BUFFER,L,PACKET,JPAKNO,LAST)
        CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
        CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
        COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK3/NPI,NPO
        CHARACTER*80 BUFFER
        CHARACTER*20 PACKET
C       WRITE(6,201)
 201    FORMAT('*** ENTERING GETPAK ***')
C       WRITE(6,20) L
 20     FORMAT ('LENGTH = ',I10)
        K1=(JPAKNO-1)*10+1
        K2=K1+9
        IF(K2.GT.L) K2=L
        PACKET=BUFFER((K1):(K2))
C       WRITE (6,21) JPAKNO,K1,K2,PACKET
 21     FORMAT ('PACKET NO.,K1,K2 ',3I20,' PACKET IS *',A10)
        IF (K2.LT.L) GOTO 100
        LAST =1
        GOTO 500
 100    CONTINUE
        LAST =0
 500    CONTINUE
C       WRITE(6,211)LAST
 211    FORMAT('IN GETPAK, LAST IS ',I8)
C       WRITE(6,43)
 43     FORMAT('*** LEAVING GETPAK ***')
        RETURN
        END
```

```
C      SUBROUTINE TO TRANSMIT A FRAME
C      INPUT : FRAME
C      OUTPUT: STATUS - 1 IF SUCCESSFUL WITHIN 6 TRIES
C                     - 0 IF NOT SUCCESSFUL WITHIN 6 TRIES
       SUBROUTINE XMTFRM (FRAME,STATUS)
       CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
       CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
       COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
       COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
       COMMON/BLK3/NPI,NPO
       CHARACTER*20 FRAME
       CHARACTER*80 STRING
       CHARACTER*80 RES
       INTEGER STATUS
C      WRITE(6,201)
 201   FORMAT('*** ENTERING XMTFRM ***')
       NTRIES=0
       STRING=FRAME
 50    CALL PWRITE(NPO,STRING,17)
       NTRIES=NTRIES+1
       NCHAR=80
C      NUMBER OF CHARACTERS TO BE READ
       CALL PREAD (NPI,RES,NCHAR,NEND)
       IF(NEND.EQ.0)GOTO 140
C      ENQ RECEIVED OUT OF SYNC - RESET TO PROCESS MODE.
       STATUS=3
       NEND=0
       GOTO 500
C      WRITE(6,205)NTRIES,RES
 205   FORMAT('NTRIES AND RES ARE ',I8,'->',A10,'<-')
C      REPLACED BY FOLLOWING 3 LINES - IF (RES.EQ.ACK) GOTO 100
 140   IFOUND=0
       IFOUND=INDEX(RES,ACK)
       IF (IFOUND.NE.0) GOTO 100
C      REPLACED BY FOLLOWING THREE LINES - IF (RES.NE.NAK) GOTO 200
       IFOUND=0
       IFOUND=INDEX(RES,NAK)
       IF (IFOUND.EQ.0) GOTO 200
       IF (NTRIES.LT.6) GOTO 50
       STRING=EOT
       CALL PWRITE(NPO,STRING,4)
       GOTO 200
C      SEND SUCCESSFUL: SET STATUS=1 AND RETURN
 100   STATUS=1
       GOTO 500
C      SEND FAILED: SET STATUS=0 AND RETURN
 200   STATUS=0
 500   CONTINUE
C      WRITE(6,43)
 43    FORMAT('*** LEAVING XMTFRM ***')
       RETURN
       END
C
```

```
C     ************************ RECEIVE ************************
C
      SUBROUTINE RECIVE(ISYSID,STRING,BUFFER,LGT,ISYSTO,STATUS)
      INTEGER STATUS,ISYSID
      CHARACTER *80 BUFFER,STRING
      CHARACTER *20 FRAME
      CHARACTER *20 PACKET
      CHARACTER*2 TYPE,CSUM
      CHARACTER*4 RES
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
C
C     WRITE(6,201)
 201  FORMAT('*** ENTERING RECIVE ***')
 80   NCOUNT=0
      LGT=0
C     'NCOUNT' COUNTS THE NUMBER OF PACKETS RECEIVED
C
C     ANALYZE THE FRAME
 85   IF(NEND.EQ.0)GOTO 86
      STATUS=3
      NEND=0
      GOTO 500
 86   FRAME=STRING(1:20)
      CALL ANALYZ(FRAME,TYPE,PACKET,CSUM,ISYSTO)
      IF((TYPE.EQ.MSG).OR.(TYPE.EQ.LSTMSG))GOTO 100
C     INVALID FRAME  - SHOW FAILURE AND QUIT
      STATUS=0
      GOTO 500
C
C     APPEND PACKET IF GOOD
 100  IF (CSUM.NE.GOOD) GOTO 110
C     GOOD CHECKSUM
      NCOUNT=NCOUNT+1
      K1=(NCOUNT-1)*10+1
      K2=K1+9
      BUFFER(K1:K2)=PACKET
C     WRITE(6,53)PACKET,BUFFER
 53   FORMAT(A10,'###',A80)
      STRING=ACK
      CALL PWRITE(NPO,STRING,4)
      IF(TYPE.NE.LSTMSG)GOTO 90
C     LAST MESSAGE-SET LENGTH,SHOW SUCCESS AND RETURN
      STATUS=1
      LGT=K2
      GOTO 500
C     BAD CHECKSUM-SEND 'NAK' AND LOOP
 110  STRING=NAK
      CALL PWRITE(NPO,STRING,4)
 90   NCHAR=80
C     NUMBER OF CHARACTERS TO BE READ
      CALL PREAD(NPI,STRING,NCHAR,NEND)
      GOTO 85
C     NO MESSAGE-SHOW FAILURE AND QUIT
 200  STATUS=2
      GOTO 500
 500  CONTINUE
```

```
C       WRITE(6,43)
 43     FORMAT('*** LEAVING RECEIVE ***')
       RETURN
       END
C
```

```
C       SUBROUTINE TO ANALYZE THE FRAME
        SUBROUTINE ANALYZ(FRAME,TYPE,PACKET,CSUM,ISYSTO)
        CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
        CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
        COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK3/NPI,NPO
        CHARACTER*20 FRAME
        CHARACTER*20 PACKET
        CHARACTER*2 CH,CTEMP,TYPE,CSUM
C       WRITE(6,40)
 40     FORMAT('*** ENTERING ANALYZE ***')
        IFOUND=0
        IFOUND=INDEX(FRAME,MSG)
        IF(IFOUND.NE.0)GOTO 100
        IFOUND=INDEX(FRAME,LSTMSG)
        IF(IFOUND.NE.0)GOTO 100
        IFOUND=1
 100    TYPE=FRAME(IFOUND:IFOUND+1)
        JFROM=ICHAR(FRAME(IFOUND+2:IFOUND+2))
        ISYSTO=ICHAR(FRAME(IFOUND+3:IFOUND+3))
        ISYSTO=ISYSTO-50
C       SINCE IT WAS SENT AS RADIX 50
C
        JC=ICHAR(FRAME(IFOUND+4:IFOUND+4))
C       WRITE(6,44)JFROM,ISYSTO,JC
 44     FORMAT(' JFROM,ISYSTO AND JC ARE ',3I6)
        PACKET=FRAME(IFOUND+5:IFOUND+14)
        CSUM=FRAME(IFOUND+15:IFOUND+16)
        CALL CHKSUM(PACKET,CH)
C       WRITE(6,41) TYPE,PACKET,CSUM,CH
 41     FORMAT(' TYPE,PACKET,CSUM,CH ARE +',A2,'+',A10,'+',A2,'+',A2)
        CTEMP=LSTMSG
        IF(CH.EQ.CSUM)CTEMP=GOOD
        CSUM=CTEMP
C       WRITE(6,43)
 43     FORMAT('*** LEAVING ANALYZE ***')
        RETURN
        END
```

```
C       SUBROUTINE TO COMPUTE THE CHECKSUM
        SUBROUTINE CHKSUM(PACKET,CH)
        CHARACTER*20 PACKET
        CHARACTER*2 CH
C       WRITE(6,32)
 32     FORMAT(' *** ENTERING CHKSUM ***')
        ISUM=0
        DO 10 I=1,10
        ISUM=ISUM+ICHAR(PACKET(I:I))
 10     CONTINUE
        I1=ISUM/10
C       WRITE(6,35)ISUM
 35     FORMAT(' ISUM IS ',I6)
        I2=ISUM-I1*10+48
        CH(1:1)=CHAR(I1)
        CH(2:2)=CHAR(I2)
 40     FORMAT (A1)
C       WRITE(6,33)CH
 33     FORMAT (' CHECKSUM IS -->',A2,'<--')
C       WRITE(6,34)
 34     FORMAT(' **** LEAVING CHKSUM ****')
        RETURN
        END
```

2. SEND

```
      SUBROUTINE SEND (ISYSID,BUFFER,NCHAR,STATUS)
C     SUBROUTINE TO SEND DATA BUFFER TO SYSTEM 'SYSID'
      CHARACTER*4 RES
      CHARACTER*80 BUFF,BUFFER
      CHARACTER*20 PACKET,PACK
      CHARACTER*20 FRAME
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
      INTEGER STATUS
      ENQ = 'ENQ*'
      PRE = 'PRE*'
      ACK = 'ACK*'
      NAK = 'NAK*'
      EOT = 'EOT*'
      NOMSG='NM'
      MSG='MS'
      LSTMSG='LM'
      GOOD='GD'
      NPI=11
      NPO=12
      NCHAR=NCHAR/5
      NCHAR=(NCHAR+1)*5
      JPAKNO=0
C     SEND THE ENQ AND WAIT FOR RESPONSE
      BUFF=ENQ
      CALL PWRITE (NPO,BUFF,4,NSTAT)
      IF (NSTAT.EQ.1) GOTO 105
      STATUS=0
      GOTO 500
 105  CALL PREAD (NPI,BUFF,4)
      RES=BUFF(1:4)
C      WRITE (6,20) RES
 20   FORMAT (A4)
      IFOUND=INDEX(BUFF,PRE)
      IF(IFOUND.NE.0) GOTO 100
C     RESPONSE NEGATIVE. INDICATE FAILURE AND RETURN
      STATUS = 0
      GOTO 500
C     RESPONSE POSITIVE. PROCEED
 100  CONTINUE
      JPAKNO=JPAKNO+1
      CALL GETPAK(BUFFER,NCHAR,PACK,JPAKNO,LAST)
      CALL CODE(PACK,PACKET)
      CALL BLDFRM(ISYSID,PACKET,LAST,FRAME)
      CALL XMTFRM(FRAME,STATUS)
      IF (STATUS.EQ.0)GOTO 500
      IF (LAST.EQ.0)GOTO 100
 500  CONTINUE
      RETURN
      END
```

```
C       SUBROUTINE TO RETURN NEXT PACKET
C       INPUT : BUFFER,PACKET NUMBER REQUIRED
C               LENGTH OF THE BUFFER
C       OUTPUT: PACKET,LAST
C       LAST=0 IF MORE PACKETS, 1 IF LAST PACKET
        SUBROUTINE GETPAK(BUFFER,L,PACKET,JPAKNO,LAST)
        CHARACTER*80 BUFFER
        CHARACTER*20 PACKET
C        WRITE (6,20) L
 20     FORMAT ('LENGTH = ',I10)
        K1=(JPAKNO-1)*5+1
        K2=K1+4
        IF(K2.GT.L) K2=L
        PACKET=BUFFER((K1):(K2))
C        WRITE (6,21) JPAKNO,K1,K2,PACKET
 21     FORMAT ('PACKET NO.,K1,K2 ',3I20,' PACKET IS *',A10)
        IF (K2.LT.L) GOTO 100
        LAST =1
        GOTO 500
 100    CONTINUE
        LAST =0
 500    CONTINUE
        RETURN
        END
```

```
C      SUBROUTINE TO TRANSMIT A FRAME
C      INPUT : FRAME
C      OUTPUT: STATUS - 1 IF SUCCESSFUL WITHIN 6 TRIES
C                     - 0 IF NOT SUCCESSFUL WITHIN 6 TRIES
       SUBROUTINE XMTFRM (FRAME,STATUS)
       CHARACTER*80 BUFF
       CHARACTER*20 FRAME
       CHARACTER*20 PACKET
       CHARACTER*4 RES
       CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
       COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
       COMMON/BLK3/NPI,NPO
       INTEGER STATUS
       NTRIES=0
 50    BUFF=FRAME
       CALL PWRITE (NPO,BUFF,17,NSTAT)
       IF (NSTAT.NE.1) GOTO 200
       NTRIES=NTRIES+1
       CALL PREAD (NPI,BUFF,4)
       IFOUND=INDEX(BUFF,ACK)
       IF (IFOUND.NE.0) GOTO 100
       IFOUND=INDEX(BUFF,NAK)
       IF (IFOUND.EQ.0) GOTO 200
       IF (NTRIES.LT.6) GOTO 50
       BUFF=EOT
       CALL PWRITE(NPO,BUFF,4,NSTAT)
       GOTO 200
C      SEND SUCCESSFUL: SET STATUS=1 AND RETURN
 100   STATUS=1
       GOTO 500
C      SEND FAILED: SET STATUS=0 AND RETURN
 200   STATUS=0
 500   CONTINUE
       RETURN
       END
C
```

```
C       SUBROUTINE TO BUILD FRAME
        SUBROUTINE BLDFRM(ISYSID,PACKET,LAST,FRAME)
C       INPUT - PACKET
C       OUTPUT - FRAME (MSG TYPE+PACKET+CHECKSUM)
        CHARACTER*20 PACKET
        CHARACTER*20 FRAME
        CHARACTER*2 CH
        CHARACTER*4 ACK,NAK,ENQ,PRE,EOT
        CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT
        COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
        IFROM=48
        IC=50
C        WRITE(6,201)
 201    FORMAT('*** ENTERING BLDFRM ***')
        FRAME(1:2)=MSG
        IF(LAST.EQ.1)FRAME(1:2)=LSTMSG
        WRITE(FRAME(3:3),100)IFROM
C       SINCE SYSTEM ID IS RADIX 50, ADD 50 TO IT
        ITO=ISYSID+50
        WRITE(FRAME(4:4),100)ITO
 100    FORMAT(A1)
        WRITE(FRAME(5:5),100)IC
        FRAME(6:15)=PACKET
        CALL CHKSUM(PACKET,CH)
        FRAME(16:17)=CH
C        WRITE (6,43)
 43     FORMAT('*** LEAVING BLDFRM ***')
        RETURN
        END
```

```
C       SYSTEM DEPENDENT SUBROUTINE TO READ 'NCHAR' CHARACTERS
C       INTO BUFFER 'BUFFER' FROM PORT 'NPORT'
        SUBROUTINE PREAD(NPORT,BUFFER,NCHAR)
        CHARACTER*80 BUFFER
        READ (NPORT,10)BUFFER
 10     FORMAT(A80)
        RETURN
        END
```

```
C       SYSTEM DEPENDENT SUBROUTINE TO WRITE ,NCHAR, CHARACTERS
C       FROM BUFFER 'BUFFER' TO PORT 'NPORT'
        SUBROUTINE PWRITE(NPORT,BUFFER,NCHAR,STATUS)
        CHARACTER*80 BUFFER
        CHARACTER*80 BUFF2
        INTEGER STATUS
        STATUS=1
        IER=0
        CALL WAIT(3,2,IER)
        IF (NCHAR.EQ.4) GOTO 104
        IF (NCHAR.EQ.17) GOTO 117
C       ERROR - RETURN
        STATUS=0
        GOTO 600
C       HANDLE VARIOUS CHARACTERS)
 104    WRITE(NPORT,204)BUFFER
 204    FORMAT(' ',A4)
        GOTO 500
 117    WRITE(NPORT,217)BUFFER
 217    FORMAT(' ',A17)
        GOTO 500
C       READ ECHO FROM THE WRITE -- PE ALWAYS ECHOES?
 500    NPORT1=NPORT-1
        READ(NPORT1,205)BUFF2
 205    FORMAT(A80)
 600    RETURN
        END
```

```
C       SUBROUTINE TO COMPUTE CHECKSUM
        SUBROUTINE CHKSUM(PACKET,CH)
        CHARACTER*20 PACKET
        CHARACTER*2 CH
C        WRITE(6,31)
 31     FORMAT(' *** ENTERING CHKSUM ***')
        ISUM=0
        DO 10 I=1,10
        ISUM=ISUM+ICHAR(PACKET(I:I))
 10     CONTINUE
        I1=ISUM/10
        I2=ISUM-I1*10+48
C        WRITE(6,32)ISUM
 32     FORMAT(' INTEGER CHECKUM IS ',I6)
        WRITE(CH(1:1),40)I1
        WRITE(CH(2:2),40)I2
 40     FORMAT(A1)
C        WRITE(6,33)CH
 33     FORMAT(' CHECKSUM IS -->',A2,'<--')
C        WRITE(6,34)
 34     FORMAT(' *** LEAVING CHECKSUM ***')
        RETURN
        END
```

```
C       SUBROUTINE TO CODE A PACKET
        SUBROUTINE CODE(PACK,PACKET)
        CHARACTER*20 PACK,PACKET
        DO 10 I=1,5
        I1=ICHAR(PACK(I:I))/16
        I2=ICHAR(PACK(I:I))-I1*16
        J=(I-1)*2+1
        IJ=32+I1
        WRITE(PACKET(J:J),40)IJ
        IJ=32+I2
        WRITE(PACKET((J+1):(J+1)),40)IJ
 40     FORMAT(A1)
 10     CONTINUE
        RETURN
        END
```

3. RECEIVE

```
C
C
      SUBROUTINE RECIVE(SYSID,BUFFER,STATUS)
      INTEGER STATUS,SYSID
      CHARACTER*80 BUFFER,BUFF
      CHARACTER*4 RES
      CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
      CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
      COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
      COMMON/BLK3/NPI,NPO
      CHARACTER*80 FRAME
      CHARACTER*20 PACKET
      CHARACTER*2 TYPE,CSUM
      ACK='ACK*'
      NAK='NAK*'
      ENQ='ENQ*'
      PRE='PRE*'
      REQ='REQ*'
      NOMSG='NM'
      MSG='MS'
      LSTMSG='LM'
      GOOD='GD'
      NPI=11
      NPO=12
C
C     ESTABLISH COMMUNICATIONS WITH THE MESSAGE SERVER
      BUFF=ENQ
      CALL PWRITE (NPO,BUFF,4,NSTAT)
      IF(NSTAT.EQ.1) GOTO 85
C     ERROR IN WRITE - RETURN
      STATUS=0
      GOTO 500
 85   CALL PREAD (NPI,BUFF,4)
      RES=BUFF(1:4)
C      WRITE(6,201)RES
 201  FORMAT(' RESPONSE TO ENQ* IS ',A20)
      IFOUND=INDEX(BUFF,PRE)
      IF(IFOUND.NE.0) GOTO 80
C     BAD RESPONSE - QUIT
      STATUS=0
      GOTO 500
C
C     GOOD RESPONSE - SEND REQUEST FRAME
 80   NCOUNT=0
C     'NCOUNT' COUNTS THE NUMBER OF PACKETS RECEIVED
      BUFF=REQ
      CALL PWRITE(NPO,BUFF,4,NSTAT)
      IF (NSTAT.EQ.1) GOTO 90
C     ERROR IN WRITE - RETURN
      STATUS=0
      GOTO 500
 90   CALL PREAD(NPI,BUFF,17)
      FRAME=BUFF
C
C     ANALYZE THE FRAME
C      WRITE(6,202)FRAME
 202  FORMAT (' THE FRAME RECEIVED WAS-->', A80)
      CALL ANALYZ(FRAME,TYPE,PACKET,CSUM)
      IF(TYPE.EQ.NOMSG)GOTO 200
```

```
      IF((TYPE.EQ.MSG).OR.(TYPE.EQ.LSTMSG))GOTO 100
C     INVALID RESPONSE - SHOW FAILURE AND QUIT
      STATUS=0
      IF(TYPE.EQ.NOMSG)STATUS=2
      GOTO 500
C
C     APPEND PACKET IF GOOD
 100  IF (CSUM.NE.GOOD) GOTO 110
C     GOOD CHECKSUM
      NCOUNT=NCOUNT+1
      K1=(NCOUNT-1)*5+1
      K2=K1+4
      BUFFER(K1:K2)=PACKET
C     WRITE(6,53)PACKET,BUFFER
 53   FORMAT(A10,'+++',A80)
      BUFF=ACK
      CALL PWRITE(NPO,BUFF,4,NSTAT)
      IF(TYPE.NE.LSTMSG)GOTO 90
C     LAST MESSAGE-SHOW SUCCESS AND RETURN
      STATUS=1
      GOTO 500
C     BAD CHECKSUM-SEND 'NAK' AND LOOP
 110  BUFF=NAK
      CALL PWRITE(NPO,BUFF,4,NSTAT)
      GOTO 90
C     NO MESSAGE-SHOW FAILURE AND QUIT
 200  STATUS=2
      GOTO 500
 500  CONTINUE
      RETURN
      END
```

```
C
C       SUBROUTINE TO ANALYZE THE FRAME
        SUBROUTINE ANALYZ(FRAME,TYPE,PACKET,CSUM)
        CHARACTER*4 ACK,NAK,ENQ,PRE,EOT,REQ
        CHARACTER*2 NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK1/ACK,NAK,ENQ,PRE,EOT,REQ
        COMMON/BLK2/NOMSG,MSG,LSTMSG,GOOD
        COMMON/BLK3/NPI,NPO
        CHARACTER*80 FRAME
        CHARACTER*20 PACKET,PACK
        CHARACTER*2 CH,CTEMP,TYPE,CSUM
C        WRITE(6,40)
 40     FORMAT('*** ENTERING ANALYZE ***')
C        WRITE(6,201)FRAME
 201    FORMAT(' THE FRAME IS->',A80)
        IFOUND=INDEX(FRAME,MSG)
        IF (IFOUND.NE.0) GOTO 100
        IFOUND=INDEX(FRAME,LSTMSG)
        IF (IFOUND.NE.0) GOTO 100
        IFOUND=INDEX(FRAME,NOMSG)
        IF (IFOUND.NE.0) GOTO 100
        IFOUND=1
100     TYPE=FRAME(IFOUND:IFOUND+1)
        JFROM=ICHAR(FRAME(IFOUND+2:IFOUND+2))
        JTO=ICHAR(FRAME(IFOUND+3:IFOUND+3))
        JC=ICHAR(FRAME(IFOUND+4:IFOUND+4))
C        WRITE(6,44)JFROM,JTO,JC
 44     FORMAT(' JFROM,JTO AND JC ARE ',3I6)
        PACK=FRAME(IFOUND+5:IFOUND+14)
        CSUM=FRAME(IFOUND+15:IFOUND+16)
        CALL CHKSUM(PACK,CH)
C        WRITE(6,41) TYPE,PACK,CSUM,CH
 41     FORMAT(' TYPE,PACK,CSUM,CH ARE +',A2,'+',A10,'+',A2,'+',A2)
        CTEMP=LSTMSG
        IF(CH.EQ.CSUM)CTEMP=GOOD
        CSUM=CTEMP
        IF(CSUM.EQ.GOOD)CALL DDCODE(PACK,PACKET)
C        WRITE(6,43)
 43     FORMAT('*** LEAVING ANALYZE ***')
        RETURN
        END
```

```
C      SYSTEM DEPENDENT SUBROUTINE TO WRITE 'NCHAR' CHARACTERS
C      FROM BUFFER 'BUFFER' TO PORT 'NPORT'
       SUBROUTINE PWRITE(NPORT,BUFFER,NCHAR,STATUS)
C      STATUS : 0 IF FAILURE, 1 IF SUCCESS
       INTEGER STATUS
       CHARACTER*80 BUFFER
       CHARACTER*80 BUFF2
       STATUS =1
       IER=0
C       WRITE(6,555)
 555   FORMAT (' *** WAIT STARTS ***')
       CALL WAIT(10,2,IER)
C       WRITE(6,556)
 556   FORMAT(' *** WAIT OVER ***')
       IF (NCHAR.EQ.4)GOTO 104
       IF (NCHAR.EQ.17)GOTO 117
C      ERROR - RETURN
       STATUS=0
       GOTO 600
 104   WRITE(NPORT,204)BUFFER
 204   FORMAT(' ',A4)
       GOTO 500
 117   WRITE(NPORT,217)BUFFER
 217   FORMAT(' ',A17)
       GOTO 500
C      GRAB ECHO FROM PE - -
 500   NPORT1=NPORT-1
       READ(NPORT1,205)BUFF2
 205   FORMAT(A80)
 600   RETURN
       END
```

```
C       SYSTEM DEPENDENT SUBROUTINE TO READ 'NCHAR' CHARACTERS
C       FROM PORT 'NPORT' INTO BUFFER 'BUFFER'
        SUBROUTINE PREAD(NPORT,BUFFER,NCHAR)
        CHARACTER*80 BUFFER
        READ(NPORT,10)BUFFER
 10     FORMAT (A80)
        RETURN
        END
```

```
C       SUBROUTINE TO COMPUTE CHECKSUM
        SUBROUTINE CHKSUM(PACKET,CH)
        CHARACTER*20 PACKET
        CHARACTER*2 CH
        ISUM=0
C        WRITE(6,31)
 31     FORMAT(' *** ENTERING CHKSUM ***')
        DO 10 I=1,10
        ISUM=ISUM+ICHAR(PACKET(I:I))
 10     CONTINUE
        I1=ISUM/10
        I2=ISUM-I1*10+48
        WRITE(CH(1:1),40)I1
        WRITE(CH(2:2),40)I2
 40     FORMAT (A1)
C        WRITE(6,32)CH
 32     FORMAT(' CHECKSUM IS -->',A2,'<--')
C        WRITE(6,33)
 33     FORMAT(' *** LEAVING CHKSUM ***')
        RETURN
        END
```

```
C       SUBROUTINE TO DECODE A PACKET
        SUBROUTINE DDCODE(PPACK,PACKET)
        CHARACTER*20 PPACK,PACKET
        DO 10 I=1,5
        J1=2*(I-1)+1
        J2=J1+1
        I1=ICHAR(PPACK(J1:J1))
        I2=ICHAR(PPACK(J2:J2))
        I1=I1-32
        I2=I2-32
        I3=16*I1+I2
        WRITE(PACKET(I:I),40)I3
 40     FORMAT(A1)
 10     CONTINUE
C        WRITE(6,42)PPACK,PACKET
 42     FORMAT(' IN DDCODE, PPACK & PACKET ARE -->',A20,'<-->',A20)
        RETURN
        END
```

BIBLIOGRAPHY

1. "Computer Networks", Andrew S. Tanenbaum, Prentice Hall Inc., 1981

2. "Principles of Computer Communications Network Design", J. Seider, John Wiley & Sons, Inc., 1983

3. "Local Networks", William Stallings, McMillan Publishing Co., 1984

4. "Local Area Networks - Possibilities for Personal Computers", H.J. Saal, 'Byte', October 1981, pp 92-112

5. "Design and Analysis of Computer Communications Networks", Vijay Ahuja, McGraw Hill Book Co. (1982)

6. "Manufacturing Automation Protocol - MAP", General Motors Publication (1984)